Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA INDUSTRIAL

ÁREA DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE MÁSTER

DISEÑO, DESARROLLO Y VALIDACIÓN DE LIBRERÍA DOMÓTICA IEC 61131-3

D. Fernández Villanueva, Pedro Jesús
TUTOR: D. Felipe Mateos Martín

FECHA: Septiembre 2020

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1.- Datos generales

Datos Generales del proyecto	
Título del Proyecto	Diseño, Desarrollo y Validación de Librería Domótica IEC 61131-3
Área	Ingeniería de Sistemas y Automática
Autor del Proyecto	Pedro Jesús Fernández Villanueva
Tutor del Proyecto	Felipe Mateos Martín

Tabla 1-1.- Descripción

1.2.- Resumen

El proyecto consiste en el diseño, desarrollo y validación de una librería domótica, con el fin de ofrecer opciones para el control de una instalación de una manera rápida y sencilla basándose en el estándar de programación IEC 61131-3 para autómatas programables industriales.

Se ha pensado en funciones típicas para una instalación domótica, inspirándose en la metodología que se sigue con ISA-88 para la creación de los distintos elementos de programación y de esa forma llegar al objetivo propuesto.

Se ha pretendido que resulte lo más modular posible, incluir visualizaciones asociadas a los elementos que se creen, y hacerlo de una forma sencilla que permita acceder a los datos para distintas tareas de monitorización o análisis posteriores.

1.3.- Estado del arte

Antes de empezar el desarrollo del proyecto se reúne información sobre los puntos más importantes que atañen al trabajo con el fin de entender y guiar los esfuerzos en la dirección

correcta. Estos puntos abarcan tanto los conceptos generales de domótica, las metodologías que se pretenden aplicar, sus características y límites de aplicación.

1.3.1.- Domótica

El término *Domótica* hace referencia al conjunto de tecnologías que permite la gestión y control de las instalaciones de una vivienda, pudiendo entre otras cosas, manejarlas de forma automática o a distancia, o aumentar la eficiencia del uso de los diferentes elementos [1].

Figura 1-1 - Concepto de domótica

Las áreas principales de actuación de la domótica suelen clasificarse como la gestión de sistemas de seguridad, gestión energética, bienestar o comunicaciones.

- **Sistemas de seguridad:** en función de los elementos presentes en la vivienda se podrían distinguir diferentes funciones de seguridad.

    - Alarmas técnicas: con detectores de gas, de inundación, de monóxido de carbono, de incendios, de humo, etc, poder generar avisos y actuar sobre elementos de la vivienda de forma pertinente.

    - Intrusión: por medio de detectores volumétricos o perimetrales, detectar a un posible intruso o al usuario, y actuar en consecuencia.

- Control de accesos mediante cámaras de vigilancia.
- Alerta médica y teleasistencia.
- Simulación de presencia: de forma automática encender o apagar luces u otros elementos de la vivienda como si alguien estuviera dentro.

- **Gestión energética:** lo más habitual al pensar en ahorro energético es sustituir los diferentes elementos presentes en la vivienda por otros que tengan un consumo menor. Si bien esto es cierto, también se puede conseguir con una gestión eficiente de esos elementos cuando se hace de forma automática sin depender del usuario. Algunos ejemplos de ello serian:
  - Gestión de luces y cargas: en función de parámetros de consumo, el estado de ciertos sensores, o a ciertas horas del día, desconexión de equipos no prioritarios.
  - Climatización: automatización de sistemas de calefacción y aire acondicionado en función de ciertas condiciones definidas por el usuario.
- **Bienestar:** todo tipo de acciones cuyo objetivo sea mejorar la comodidad en una vivienda:
  - Automatización: distintos sistemas e instalaciones susceptibles de ser automatizados para dotarlos de un control eficiente y con fácil manejo.
  - Iluminación: como manejo de varias luces a través de un solo botón, regulación de intensidad luminosa y apagado automático.
  - Interfaces de usuario para un control sencillo.
- **Comunicaciones:** si la vivienda cuenta con la infraestructura adecuada es posible generar avisos o actuar a distancia sobre otros sistemas.
  - Aviso de alarmas.

- Control remoto a través de Internet.
- Control a través de telemandos.
- Videoporteros.
- Informes de consumo y costes.

- **Accesibilidad:** también es posible adaptar el sistema para tener en cuenta necesidades de cualquier tipo, como para personas con diferentes niveles de capacidad, independientemente de su condición de enfermedad, discapacidad o envejecimiento. Las aplicaciones en este campo tienen entonces como objetivo favorecer la autonomía personal, algunos ejemplos serían:
  - Vigilancia de lugares inaccesibles para esa persona de forma remota.
  - El registro y control de los consumos de servicios de agua, gas, o climatización en tiempo real.
  - Emitir mensajes de emergencia o activar alarmas en caso de ser necesario.
  - La transmisión constante de información entre esa persona y sus familiares o cuidadores.

1.3.2.- Arquitecturas

Para llegar a realizar las funciones anteriores, las instalaciones necesitan los elementos conocidos como controladores, sensores y actuadores.

La cantidad de control que puedan ejercer ciertos elementos, su organización, y forma de comunicación pueden diferir de una instalación a otra, de este modo, se pueden distinguir 4 arquitecturas básicas.

- **Centralizada:** en esta arquitectura, el controlador es la parte central del sistema, el que, en función de la información captada por los sensores, ejecuta la programación pertinente, y ordena funcionar a los actuadores de cierta forma.

El principal inconveniente de esta arquitectura es que si el controlador por alguna razón deja de funcionar lo hace toda la instalación también.

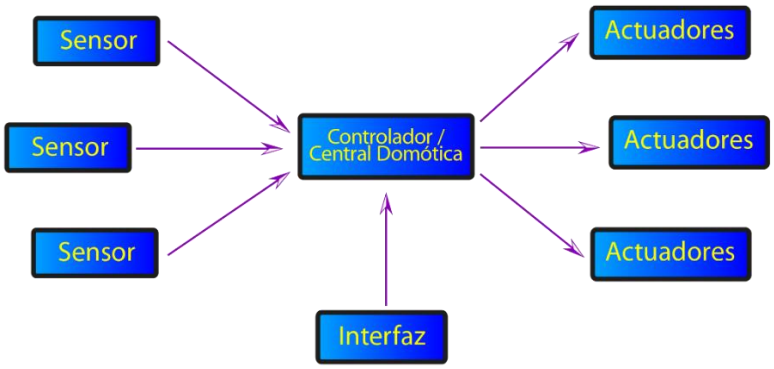

Figura 1-2 - Arquitectura centralizada

- **Periferia de descentralizada:** se trata de una variación de la centralizada, cuenta con un solo controlador, pero no se le conectan directamente los sensores y actuadores, si no que estas se reúnen en módulos de interfaz entrada salida. Esta arquitectura permite una instalación más organizada y con menos cableado, aunque sigue teniendo el mismo inconveniente que la centralizada.

- **Descentralizada:** esta arquitectura cuenta con varias unidades de control unidas mediante un bus a través del cual se comunican. Cada una de estas unidades de control se encarga de una parte de la instalación con sus propios sensores y actuadores, esto reduce los inconvenientes de las arquitecturas anteriores, pero aumenta la complejidad de las comunicaciones.

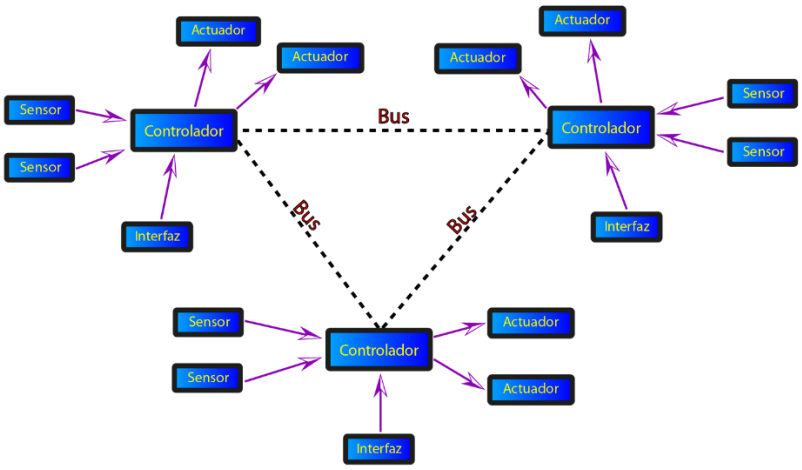

Figura 1-3 - Arquitectura descentralizada

- **Distribuida o totalmente descentralizada**: en esta arquitectura no se encuentran unidades de control como tal, sino que cada sensor y actuador posee cierta capacidad de programación con la que puede comunicarse a través de un bus con el resto de la instalación. Cada elemento posee "inteligencia propia", y la complejidad de las comunicaciones aumenta más respecto a las anteriores.

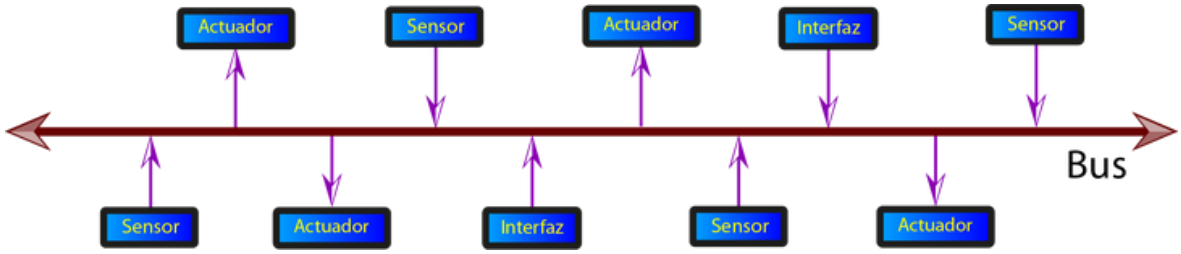

Figura 1-4 - Arquitectura distribuida

- **Mixta:** a los sistemas domóticos con estas arquitecturas también se les llama híbridos, y son combinación de las arquitecturas anteriores. Pueden tener uno o varios controladores que se comuniquen a través de un bus, y a la vez elementos con "inteligencia propia" que también utilicen ese bus y cuya información no deba pasar por los controladores.

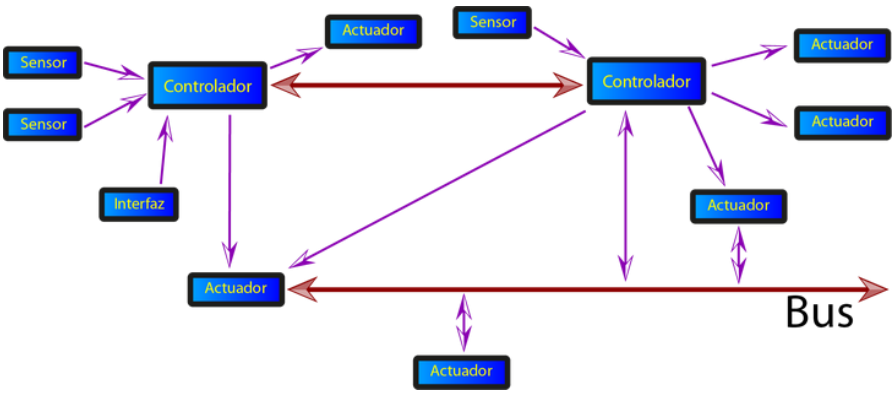

Figura 1-5 - Arquitectura mixta

1.3.3.- PLC

Un PLC (*Programmable Logic Controller*), o como se le conoce en español, un autómata programable, es un dispositivo utilizado en la automatización de sistemas, ya sea en una industria o en una vivienda particular [2].

A diferencia de las computadoras tradicionales, un PLC está diseñado para varias señales de salida y entrada, inmunidad al ruido eléctrico y resistencia a vibraciones e impactos, pudiendo alterar su programación para la función que se pretenda realizar.

1.3.3.1.- Estructura interna

Un PLC está compuesto por las interfaces de entrada y salida, y una CPU, la unidad central de procesamiento, que a su vez está compuesta por el procesador y la memoria.

La programación se encuentra almacenada en la memoria y esta es ejecutada por el procesador, que se comunica con el exterior a través de las interfaces, que transforman las señales eléctricas de los sensores (interruptores, finales de carrera, botoneras, etc) en algo que el procesador pueda interpretar, y las órdenes del procesador en señales que los actuadores (motores, luces, cargas, etc) puedan utilizar.

1.3.3.2.- Ventajas y desventajas

Algunas de las ventajas de los PLC, es que una vez instalados, las modificaciones de funcionamiento del sistema se pueden hacer únicamente alterando la programación sin costes adicionales. También suelen ser de tamaño reducido y fácil mantenimiento.

La principal desventaja, es que, para poder realizar esos cambios en la programación, o realizar el mantenimiento, se necesita a alguien especializado que conozca la instalación y funcionamiento de la programación.

1.3.4.- ISA 88

Se trata de un estándar industrial que ofrece un conjunto de normas y terminologías para control de lotes y procesos repetitivos. Algunos de los problemas que intenta abordar son la falta de un modelo universal para el control de procesos y la dificultad en transmitir las necesidades del usuario, para finalmente poder ofrecer una metodología con la que se separe el proceso y el equipo de control. De esta forma pretende permitir usar el mismo equipo para diferentes propósitos, o realizar la misma tarea con diferentes equipos.

Para hacerlo, la norma organiza la información desde 3 perspectivas diferentes:

- Modelo de control de procedimiento: representa el equipo orientado a las acciones en una secuencia ordenada.
- Modelo físico: organiza de forma jerárquica los elementos físicos del proceso.
- Modelo de proceso: organiza de forma jerárquica las funciones que se llevan a cabo

Estos 3 modelos se combinan como se muestra en la siguiente figura, aunque dependiendo de la complejidad del proceso que describe pueden aumentar o disminuir mientras sigan siendo consistentes.

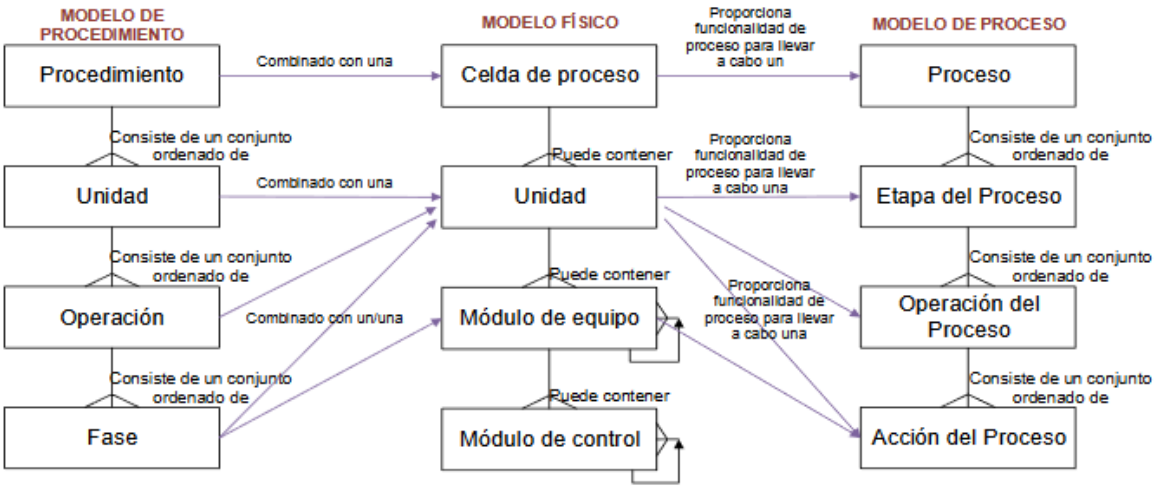

Figura 1-6 - Relación de modelos en ISA-88 [3]

Esto se aprovecha más tarde para los procesos industriales en la creación de recetas, documentación que provee la información necesaria para la manufactura de algún producto en específico.

Debido a que este proyecto no se trata de un proceso industrial como tal, pero si se basará en tecnologías de control con PLC, la metodología que ISA-88 ofrece no resulta del todo aplicable. Sin embargo, cierta parte sí podría utilizarse a fin de guiar los esfuerzos de programación como los modelos de control, que se basan únicamente en software.

1.3.5.- IEC 61131-3

IEC 61131-3 es una de las 8 partes de un estándar internacional diseñado para Controladores Lógicos Programables (PLC) y sus periféricos [4].

Este es resultado del esfuerzo de 7 multinacionales y otras entidades públicas y privadas que añaden sus muchos años de experiencia en el campo de la automatización industrial, con aproximadamente 200 páginas de texto y más de 60 tablas, para ofrecer un modelo y lenguaje comunes.

Una buena forma de ver el estándar es dividirlo en dos partes, elementos comunes y lenguajes de programación.

1.3.5.1.- Elementos comunes

- **Tipos de datos:** para evitar errores iniciales se pueden atribuir distintos tipos a los datos, como tipo booleano, numero entero o real, byte, cadena de caracteres, fecha y hora. El usuario también puede crear tipos de datos personalizados, conocidos como derivados.

- **Variables:** permiten identificar los datos cuyo contenido puede cambiar, como entradas, salidas y partes de memoria del autómata. Al declararlas se les asigna un tipo de dato común o derivado y un valor inicial que se puede configurar. Normalmente, las variables están ligadas con una extensión al lugar donde son declaradas de forma local, esto implica que se podría utilizar el mismo nombre de variable en distintos puntos del proyecto, con distintos usos, evitando errores derivados de ello. También si se prefiere es posible declarar la variable como global y que sea única en todo el proyecto.

- **Configuración, recursos y tareas:** Se establece una estructura por "capas", de más general a más específica, para las soluciones de control. Estas serían las siguientes:

  - **Configuración:** el elemento software requerido para solucionar un problema de control. Es específica para cada sistema incluyendo las características del hardware ya sean procesadores o direccionamiento de entradas y salidas.

  - **Recursos:** pudiendo haber uno o más dentro de las configuraciones, se pueden entender como un procesador que puede ejecutar un programa IEC.

  - **Tareas:** dentro de los recursos, tratarían de ser la ejecución de programas y/o bloques de función, pudiendo ejecutarse de forma periódica o por señales de disparo.

  - **Programas**: están formados a partir de un número variable de elementos de software, escritos en uno o más de los diferentes lenguajes de programación definidos por el IEC 61131-3.

- **Funciones y bloques funcionales:** siendo las unidades más básicas puede haber multitud de ellas en un programa, conteniendo declaraciones de variables e instrucciones, las **Unidades de Organización de Programa,** también conocidas en inglés como POU (*Program Organization Unit*), son estos bloques y funciones.

    - **Funciones:** el IEC 61131-3 especifica funciones estándar y funciones que define el usuario. Algunas de las funciones estándar son: sumar (ADD), valor absoluto (ABS) y raíz cuadrada (SQRT). Las funciones que haga el usuario pueden ser utilizadas más tarde en cualquier otro POU. La particularidad que tienen las funciones es después de utilizarse se reinician por completo, por este motivo no deben ser utilizadas con la esperanza de guardar algún valor de estado del proceso, pues se perderá al terminar.

    - **Bloques funcionales:** contienen variables y datos en una estructura de caja negra, con una interfaz de entradas y salidas definida y un código interno oculto, de este modo se establece una separación clara entre niveles de programadores. Una vez creados, es posible usarlos una y otra vez creando copias de un mismo bloque, llamadas instancias, lo que los convierte en elementos muy reutilizables. A diferencia de las funciones los bloques funcionales retienen la información sobre sus variables entre llamado y llamado.

    - **Programas:** son un conjunto lógico de elementos y construcciones con el lenguaje de programación para el tratamiento de una señal, y el control de un proceso con el autómata programable, que resultan de una combinación de variables, funciones y bloques funcionales.

1.3.5.2.- Lenguajes de programación

El estándar describe 5 lenguajes de programación, con una gramática y una sintaxis definida para evitar variaciones, los cuales serían:

- Literales
    - Lista de instrucciones (IL, *Instruction List*)
    - Texto estructurado (ST, *Structured Text*)
- Gráficos
    - Diagrama de contactos (LD, *Ladder Diagram*)
    - Diagrama de bloques funcionales (FBD, *Function Block Diagram*)
    - Gráfico funcional secuencial (SFC, *Sequential Function Chart*)

El estándar interrelaciona y permite el uso y combinación de cualquiera de los lenguajes para un programa, adecuándose a la experiencia del programador, la estructura de control, el tipo de proceso, etc.

1.4.- Alternativas para el desarrollo

Con la idea de los objetivos que se pretenden alcanzar y que filosofía utilizar, surge la cuestión de cómo se puede llevar a cabo. Para ello se buscan plataformas software a través de internet pensadas para el desarrollo de programación para PLCs, a fin de encontrar el más adecuado para el proyecto. La gran mayoría de las opciones que aparecen son programas propios de empresas, es decir, están desarrollados por ellas y diseñados para sus propios productos por lo que presentan inconvenientes para tener un carácter universal y flexible.

Algunos ejemplos de ellos serían **TIA Portal** [5]**,** un software de pago de la compañía Siemens que puede simular y desarrollar programas en varios lenguajes, o **Unilogic** [6]**,** un software de gratuito de la compañía Unitronics para la configuración y comunicación de sus PLC. Existen otros muchos con características similares…

1.4.1.- Herramienta software seleccionada

La mayoría de las plataformas revisadas para el proyecto siguen la tendencia vista hasta el momento, softwares preparados para los PLC propios de una compañía, enfocados a crear programas para una instalación ya definida que se vaya a poner en marcha.

Para este trabajo, cuyo objetivo es crear algo más general y abierto, algo que se pueda utilizar como base en otros proyectos y desarrollar programas más complejos, se ha elegido como herramienta **Codesys**, al permitir programar de una forma más general, para una amplia variedad de PLCs, contar con bastante información sobre su uso y permitir un modo de simulación y visualización intuitivos y fáciles de usar.

2. Programa Codesys

Se trata de una plataforma software gratuita orientada a la automatización industrial para una amplia variedad de PLCs [7], la cual se ha podido utilizar en alguna ocasión durante los estudios de grado, aunque en una versión anterior. Utiliza el estándar IEC 61131-3 y dispone de un modo de simulación bastante potente y de la capacidad de crear visualizaciones (interfaces de usuario), muy adecuado para los objetivos de este proyecto. Se utiliza la versión más reciente del programa en el momento de la realización del proyecto, la **versión 3.5 SP15 Patch 4.**

Figura 2-1 - Acerca de Codesys

Una vez elegida la plataforma en la que desarrollar el trabajo, se procede con explicaciones breves a fin a familiarizarse con el entorno, desde la pantalla principal, crear proyectos nuevos, elementos, visualizaciones, cómo funciona el modo de simulación, etc.

2.1.- Pantalla principal

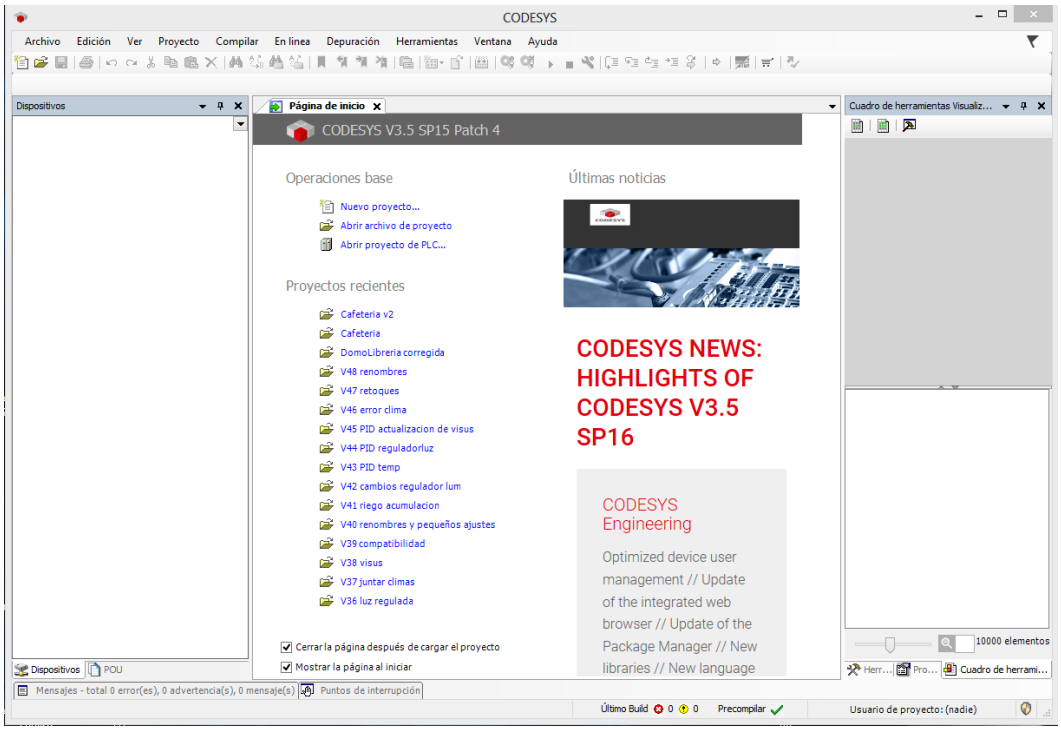

Figura 2-2 - Vista general de Codesys

La primera vez que se abre el programa se encuentra una ventana como la que se ve en la figura anterior. En la parte superior del programa se encuentra como en muchos otros la barra de menús, con diferentes opciones para su manejo.

El resto de la ventana sería el área de trabajo, la cual empieza con 4 pestañas abiertas por defecto, las cuales pueden desplazarse, cerrarse y minimizarse según se necesite. Sus funciones serian:

- **Dispositivos:** en esta ventana se ve el contenido del proyecto que este en ese momento abierto, sus componentes como bloques de funciones, programas visualizaciones, etc.

- **Página de inicio:** se encuentra al inicio del programa, donde se ven algunas opciones generales y proyectos abiertos recientemente.

En el momento que se abra un proyecto, la pestaña cambiara al área central de trabajo, donde se verán los contenidos de los diferentes elementos añadidos en la pestaña de dispositivos.

- **Mensajes:** como su nombre indica en esta pestaña se ven mensajes del programa, desde errores que Codesys detecte en la programación, advertencias por elementos mal configurados, hasta mensajes de compilación como el tamaño del programa generado.

- **Propiedades:** por último, esta pestaña presenta diferentes contenidos dependiendo de lo que se esté modificando en el área de trabajo en ese momento.

  Puede presentar desde elementos que se pueden colocar en las visualizaciones y su configuración, hasta elementos para añadir a los códigos gráficos.

2.1.1.- Creación de un proyecto

El primer paso para la creación de un proyecto comienza por la localizar en la barra de menús "*Archivo→Nuevo proyecto…*" lo cual abre una ventana de opciones como se muestra en la figura siguiente.

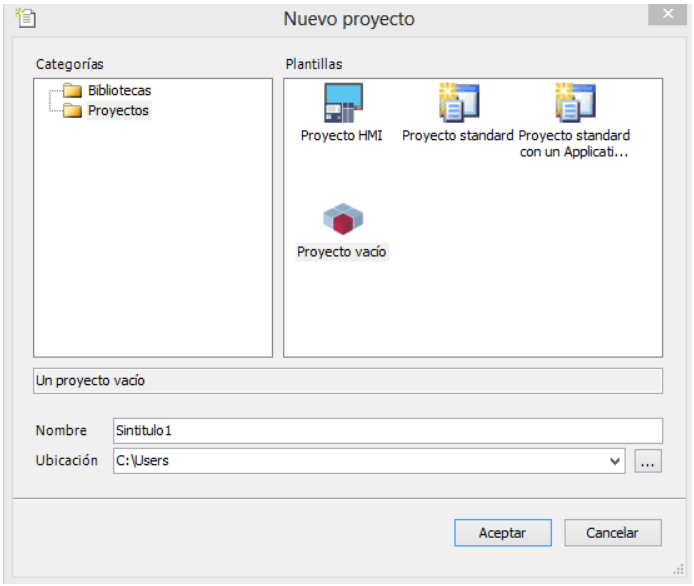

Figura 2-3 - Opciones de creación de proyecto

En esta ventana se puede elegir donde guardar el proyecto, con que nombre, si tendrá alguna configuración por defecto y si será un proyecto o una biblioteca. La configuración para el proyecto fue la de proyecto estándar, la cual abre otra ventana donde se elige el dispositivo de control que se usará en la simulación, y en que lenguaje de programación por defecto estará el programa utilizado.

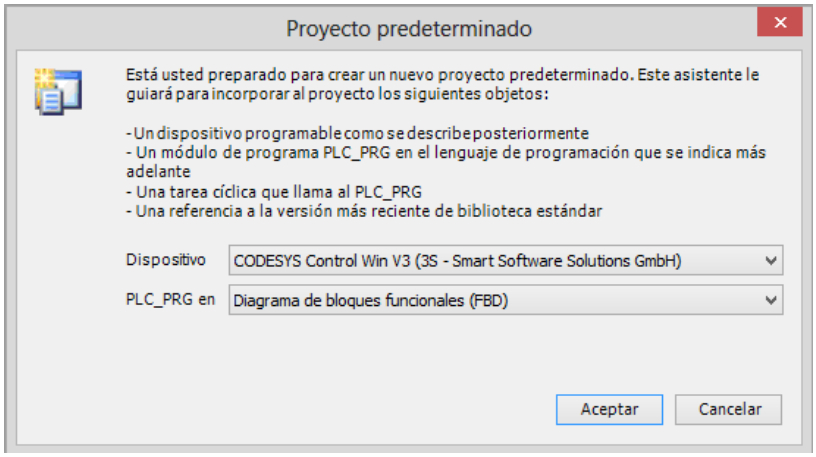

Figura 2-4 - Configuración de proyecto

Después de aceptar, se encuentra la pestaña de dispositivos como se ve en la siguiente figura, dependiendo de la opción elegida variaran los elementos presentes por defecto, desde el dispositivo a la configuración de tareas. La figura siguiente corresponde a los elementos que aparecen por defecto al elegir un proyecto estándar.

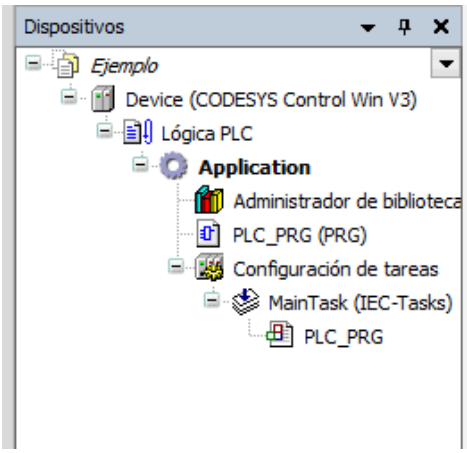

Figura 2-5 - Elementos de un proyecto estándar

Los elementos visibles más importantes serian:

- **Device:** el dispositivo elegido a programar, este se podrá conectar físicamente al ordenador para cargar el programa, o simularse.

- **Application:** sería el equivalente a las Tareas vistas en la descripción del estándar IEC 61131-3, a la cual se le añaden funciones, programas y bloques funcionales.

- **Administrador de bibliotecas:** un asistente cuyo propósito es el de añadir bibliotecas al proyecto, que ya cuenten con elementos predefinidos para ayudar en la creación de nuevos programas o visualizaciones.

- **Configuración de tareas:** sería el lugar donde colocar los programas que ejecutará el PLC una vez se le cargue el proyecto, por defecto *main task* (tarea principal) tiene ya colocado el programa, el cual se puede configurar arriba.

Para añadir otros elementos al proyecto, con el clic derecho del ratón en *Application* se pueden abrir las opciones para ello, desde agregar carpetas para poder disponer todo de una forma ordenada, a agregar objetos, donde se encuentran los POU, listas de variables y visualizaciones.

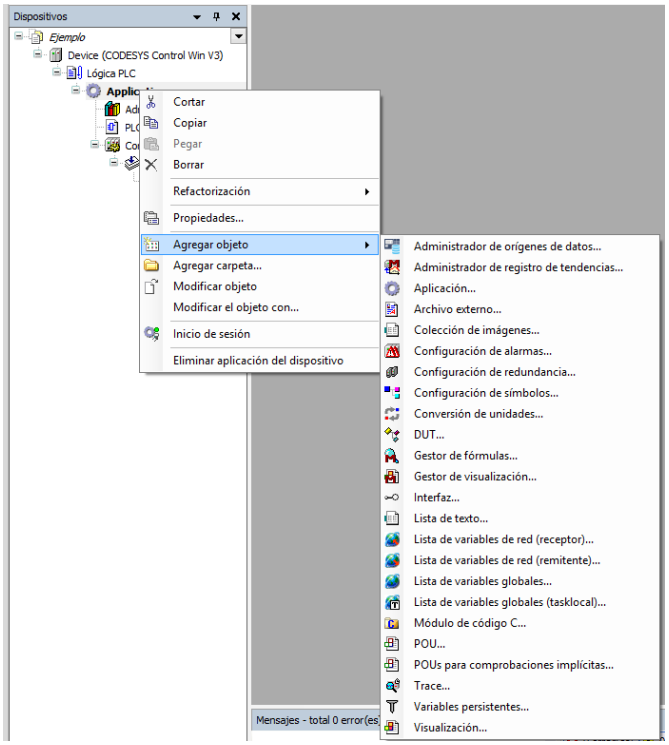

Figura 2-6 - Agregar objetos a un proyecto

A partir de aquí queda a discreción del programador como crear su aplicación.

2.2.- Creación de una biblioteca

Codesys permite agregar bibliotecas a los proyectos, y a su vez también crearlas, con ciertas limitaciones a los elementos que se pueden incluir en ellas. A la hora de crear una nueva biblioteca se sigue un proceso parecido al de un proyecto, pero esta vez después de elegir "*Archivo→Nuevo proyecto…*", en la primera ventana de configuración se elige la opción de biblioteca [8].

Al igual que los proyectos, también existen varias opciones de preconfiguración, según el tipo de biblioteca que se pretenda crear, para este caso se utilizará una Biblioteca Codesys.

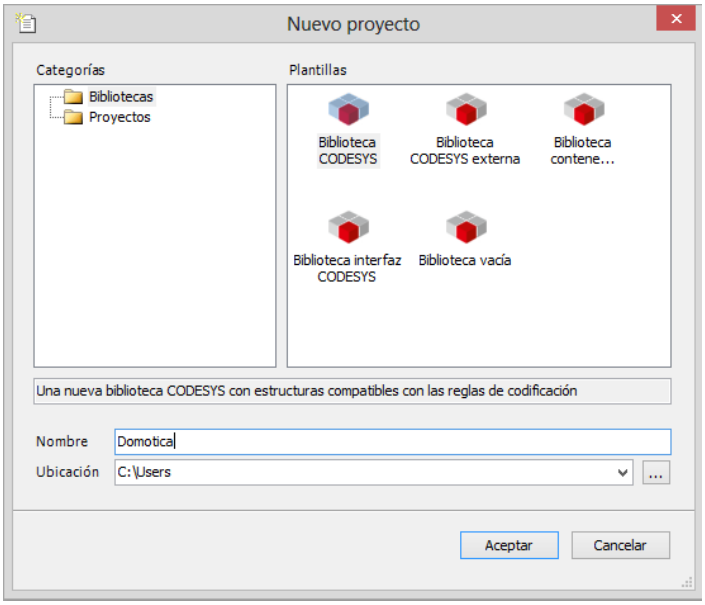

Figura 2-7 – Creación de biblioteca

Al aceptar se abre directamente lo que parece ser un proyecto vacío, pero las bibliotecas a diferencia de los proyectos tienen sus elementos en otra subpestaña de la pestaña dispositivos, concretamente en la denominada POU.

Se aconseja que una de las primeras tareas por hacer, sea la de rellenar la información del proyecto antes de hacer nada más. Esto es debido a que, si se añade una biblioteca al repositorio, y luego se altera la información del proyecto, la anterior no se borra y se crea un duplicado. Por tanto, para evitar añadir múltiples copias al repositorio y mantener al mínimo posibles confusiones a continuación se describen los datos más importantes de esta tarea.

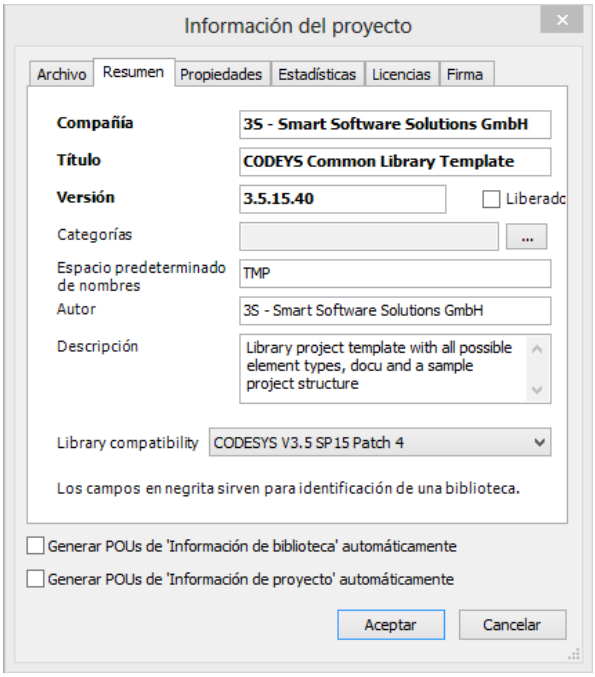

Figura 2-8 - Información de biblioteca

- **Compañía:** compañía desarrolladora de la biblioteca.
- **Título:** nombre de la biblioteca cuando se vea en el buscador del administrador de bibliotecas.
- **Versión:** indica en que versión del programa que ha sido creada o a cuál va destinada.
- **Categorías:** se le puede asignar a una categoría concreta para una mejor clasificación, haciendo referencia al contenido que tiene la biblioteca.
- **Espacio predeterminado de nombre:** esta será la palabra o abreviatura que habrá que colocar para crear instancias de los bloques que contenga la biblioteca.
- **Compañía:** autor de la biblioteca.
- **Descripción:** un espacio para texto que se verá al seleccionar la biblioteca donde se puede incluir una breve explicación del contenido de la biblioteca.

Otro paso necesario es modificar algunas de las propiedades de la biblioteca en la próxima pestaña. La más importante sería el *DocFormat*, el cual debería siempre tener el valor de texto reestructurado, para hacerlo compatible con la mayoría de los proyectos que vayan a utilizar más adelante esta biblioteca.

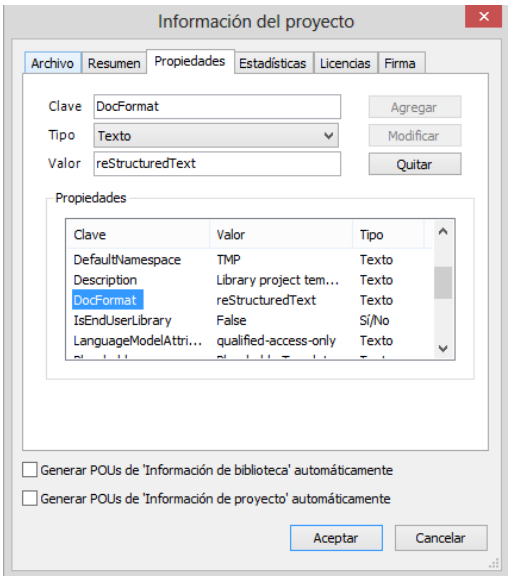

Figura 2-9 - Propiedades de biblioteca

A partir de este momento la biblioteca esta lista para empezar a crearse, con las siguientes restricciones:

- **POUS:** solamente serán visibles los bloques funcionales cuando se utilice la biblioteca en otro proyecto, las funciones y programas pueden seguir en funcionamiento internamente, pero otro usuario no podrá verlos o utilizarlos de forma directa. Los POUS se puede copiar desde un proyecto donde se hayan podido simular y probar, y pegarlos directamente, de este modo no se crea ninguna funcionalidad a ciegas.

- **Visualizaciones:** es posible añadir visualizaciones de la misma forma que se hace copiando POUS enteros, pero sus elementos no podrán contener variables o deberán tener variables auxiliares en una configuración específica para no generar errores. Si se opta por la segunda opción y se hace correctamente, más adelante Codesys

preguntara al usuario con que variable relacionar cada elemento cuando quiera utilizar una visualización de la biblioteca.

- **Otras bibliotecas:** es posible que se necesite la ayuda de alguna otra biblioteca a la hora de crear alguna visualización u otra funcionalidad, la cual también debe cargarse a través del administrador de bibliotecas de la misma forma que se hace con los proyectos para funcionar correctamente.

A partir de este momento queda a discreción del programador el crear su propia biblioteca.

Una vez finalizada, para poder utilizarla en otros proyectos, es necesario abrir el archivo que contiene la biblioteca, y utilizar el botón con forma de archivador amarillo, que está junto al icono de guardar. De esta forma tras un breve periodo de carga y si no existen fallos, la biblioteca quedará añadida al repositorio y podrá utilizarse en otros proyectos.

Figura 2-10 - Añadir al repositorio

3. Diseño y desarrollo

3.1.- Especificaciones

Como primer paso para la creación del proyecto, se toma en consideración todo lo descrito sobre domótica en los apartados anteriores, a fin de pensar en una serie de funcionalidades susceptibles de añadirse.

Sabiendo las limitaciones de elementos que luego serán visibles en una biblioteca de Codesys, se enfocará el proyecto hacia la creación de bloques funcionales, adaptando la filosofía de ISA-88 para crear sus funcionalidades. Sobre el proyecto, como ya estipuló en el apartado 1.3.4, al no tratarse de un proceso industrial sería necesario adaptar de cierto modo los modelos, y dado que únicamente tratará sobre control en software, sin equipos físicos, se toma como ejemplo el modelo de control de proceso para esquematizar como secuenciar las funciones que se desarrollen.

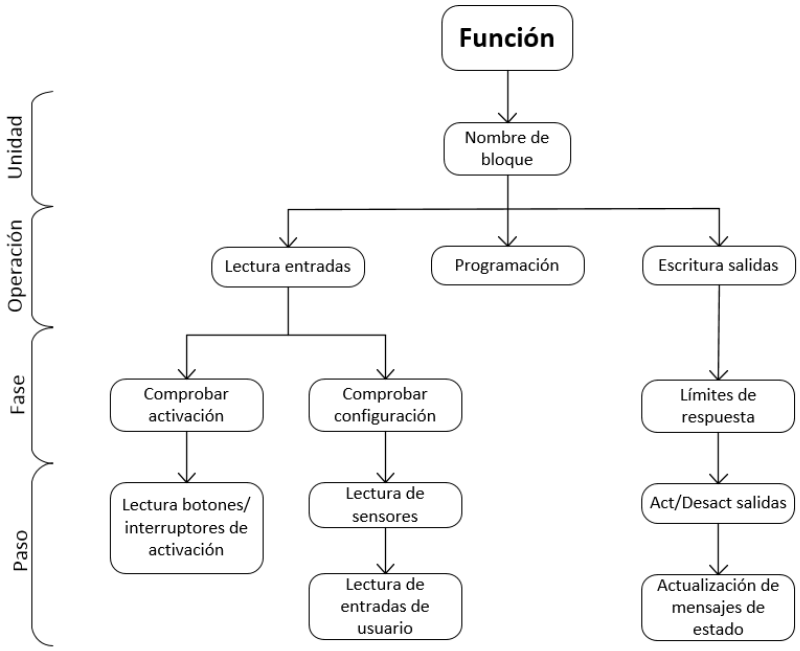

Figura 3-1 - Adaptación de ISA-88

El esquema de la figura anterior será de más o menos utilidad ya que la estructura de los bloques variará según la función que desarrollen, pero se utiliza como plantilla para desarrollar el código en adelante.

En lo que al estándar IEC 61131-3 se refiere, se utiliza directamente ya que la plataforma en la que se desarrolla el proyecto lo utiliza como base en su programación.

Por último, para demostrar la validez de la biblioteca una vez terminada, se crea una simulación como un proyecto aparte desde cero, en la que se utiliza la biblioteca como base. De esta forma se pretende demostrar su uso y funcionamiento, pudiendo crear un proyecto funcional con necesidades de programación mínimas.

3.1.1.- Funciones a incluir

Después de varias consideraciones y experimentar con el programa para familiarizarse con su funcionamiento, se piensa en añadir en un principio, funciones para gestionar los siguientes campos:

- Gestión de luces y/o cargas.
- Gestión de alarmas
- Control de sistemas motorizados
- Gestión de riego
- Sistema de climatización

Esas serían las 5 áreas en las que se ha enfocado como añadir funcionalidades para el proyecto, y a lo largo de su desarrollo se hacen más complejas, añadiendo por ejemplo controles horarios y modos automáticos.

Además, para su desarrollo se tuvieron presentes los sistemas de control comerciales para luces, climatización, etc, para así reproducir en la medida de lo posible, esas características e incluirlas en el proyecto.

En los siguientes apartados se describen cada uno de los bloques funcionales que se han diseñado, desde una descripción de su funcionamiento, entradas y salidas hasta una breve introducción al desarrollo de código realizado.

3.1.2.- Notación

A fin de seguir una metodología por la gran cantidad de entradas, salidas y variables presentes en toda la programación y poder identificar su uso o tipo más fácilmente, se ha pensado en seguir las siguientes pautas.

3.1.2.1.- Entradas y salidas

Una sencilla norma para identificarlas más fácilmente.

- **"In_":** se colocará este prefijo como identificación de que esa variable es un parámetro de entrada a un bloque funcional.
- **"Out_":** este prefijo se añade como identificación de que esa variable es un parámetro de salida de un bloque funcional.

3.1.2.2.- Tipo de variable

Como es conocido en el estándar IEC 61131-3, existen diferentes tipos de variable utilizables. Codesys cuenta con una página web de ayuda al desarrollo, donde se aconseja con que notación identificar el tipo de variable utilizado añadiendo una letra o combinación de letras al inicio de las variables. Siguiendo esta guía, se han utilizado las siguientes:

- **"x":** identifica una variable booleana, valores TRUE o FALSE.
- **"i":** identifica una variable de tipo entero, números como 0, 1, 2, 3 etc incluyendo negativos.
- **"r":** identifica una variable de tipo real, incluyendo números con decimales.
- **"t":** identifica una variable de tipo tiempo, utilizada por el programa en ciertas funciones.

- **"s":** identifica una variable de tipo *"STRING",* son variables para almacenar texto.
- **"dt":** identifica una variable tipo *"DATE AND TIME",* fecha y hora, variable que incluye el año, mes, día, hora, minuto y segundo.
- **"dat":** identifica una variable tipo "*DATE"*, solo fecha, esta solo incluye el año, mes y día.
- **"tod":** identifica una variable tipo "*TIME OF DAY"*, tiempo del día, variable con formato de solo hora, minuto y segundo.

3.1.3.- Configuración de la biblioteca

Como se vio en el apartado 2.2, la biblioteca que se cree necesita configurarse previa a su uso para crear instancias de sus bloques y poder encontrarla en el repositorio. De este modo se cubre la información pertinente, dándole la categoría de *Application* en el repositorio, y la palabra *Domotica* para crear las instancias de sus bloques, además de los nombres del autor compañía y demás parámetros.

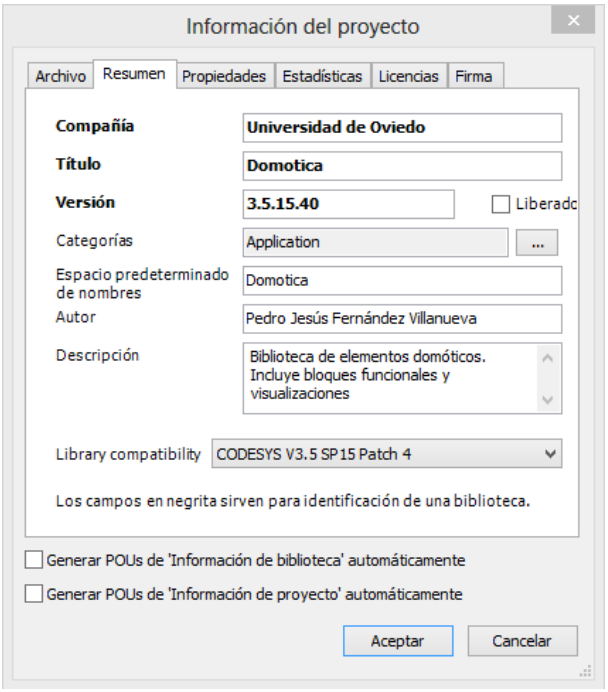

Figura 3-2 - Configuración de la biblioteca

3.1.4.- Documentación de la biblioteca

Las bibliotecas de Codesys pueden generar de forma automática documentación sobre sus contenidos, como los bloques funcionales y visualizaciones. En lo que a bloques funcionales se refiere, puede generar esquemas de entradas y salidas con los que se pueden visualizar las mismas sobre un bloque de forma gráfica.

Además, si se ha incluido algún tipo de comentario en la declaración de variables de los bloques, dependiendo de donde esté colocado, este se incluye en la documentación de forma automática. Por lo tanto, se aprovecha esta funcionalidad para incluir un resumen del funcionamiento de cada uno de los bloques y una descripción de cada una de las entradas y salidas que pueden consultarse al revisar la documentación de la biblioteca.

Cuando la biblioteca esté incluida en un proyecto se podrá acceder a esta documentación desde el administrador de bibliotecas, donde seleccionando esta biblioteca se pueden ver todos los elementos que incluye con su documentación.

3.2.- Bloque funcional de control de luces

Con este bloque funcional se pretende conseguir el manejo de cargas de iluminación según ciertas configuraciones que el usuario pueda modificar. Algunos usos comerciales encontrados han sido la regulación de luminosidad y la programación horaria.

El bloque se configura a través de un selector, con el que se puede elegir la funcionalidad a utilizar entre las 7 que lleva incluidas, con un mensaje que indica el modo en uso. Estos modos en orden son los siguientes:

1. **Interruptor:** un comportamiento sencillo y básico de cualquier instalación, la salida toma el mismo valor que la entrada.

2. **Pulsador:** alternativa al interruptor en caso de no disponer de ellos en la instalación.

3. **Detector de presencia:** mantiene la salida activa mientras la entrada esté activa, y al desactivarse la entrada hasta la desconexión de la salida. Este tiempo se puede alterar.

4. **2 salidas:** manejo de 2 cargas a través de un mismo pulsador en función del tiempo que se mantenga presionado, en la descripción del código se detalla su funcionamiento.

5. **Programación horaria:** modo automático para el manejo de la carga en función de un rango de horas definido, desde inicio a finalización que no depende de la entrada.

6. **Regulación:** salida regulable hacia un balasto en valor 0-30000, para el manejo de la intensidad luminosa de la estancia, sin sensor de luminosidad asociado.

7. **Regulación:** salida regulable hacia un balasto en valor 0-30000, para el manejo de la intensidad luminosa de la estancia, con sensor de luminosidad asociado.

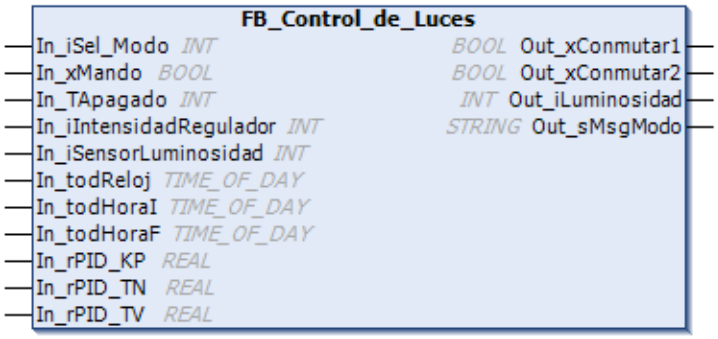

Figura 3-3 - Representación del bloque funcional de control de luces

3.2.1.- Entradas/Salidas

Variable	Descripción
In_iSel_Modo	Entrada para elegir el modo de funcionamiento. Interruptor por defecto
In_xMando	Entrada de control, interruptor, pulsador, detector
In_Tapagado	Tiempo de apagado del modo detector de presencia, por defecto 5s
In_iIntensidadRegulador	Entrada para la intensidad luminosa deseada por el usuario
In_iSensorLuminosidad	Entrada para un sensor de luminosidad
In_todReloj	Entrada para la hora actual del funcionamiento horario
In_todHoraI	Entrada para hora de inicio
In_todHoraF	Entrada para hora de finalización
In_rPID_KP	Entrada para configuración de acción proporcional
In_rPID_TN	Entrada para configuración de acción integral
In_rPID_TV	Entrada para configuración de acción derivativa
Out_xConmutar1	Primera salida, activa para todos los modos

Out_xConmutar2	Segunda salida, solo manejable con el modo de comportamiento 3
Out_iLuminosidad	Salida para intensidad de iluminación
Out_sMSGModo	Salida de texto para mostrar el modo de funcionamiento

Tabla 3-1 - E/S bloque luces

3.2.2.- Código

El funcionamiento del bloque comienza por revisar el estado de la variable de entrada *In_iSel_Modo* pudiendo tomar valores desde 1 a 7. En función del valor se ejecutará la programación de uno de los modos, indicando con la variable *Out_sMSGModo* en qué modo de funcionamiento de encuentra el bloque en ese momento.

- ***In_iSel_Modo=1:*** Modo interruptor, el más simple todos, iguala el valor de TRUE o FALSE recibido por *In_xMando* a la salida *Out_xConmutar1*, en resumen, pone la salida en el mismo estado que la entrada.

```
IF In_iSel_Modo=1 THEN

    Out_sMsgModo:='Interruptor';
    Out_xConmutar1:=In_xMando;
```

Figura 3-4 - Código interruptor

- ***In_iSel_Modo=2:*** Modo pulsador, utilizando la variable auxiliar *iContador_Pulsador*, que cambia a la vez que los estados de la variable *In_xMando*, se consigue que solo se actúe sobre la salida cuando se ha producido todo el ciclo de apretar y soltar el pulsador correspondiente.

```
ELSIF In_iSel_Modo=2 THEN

    Out_sMsgModo:='Pulsador';

    IF In_xMando=TRUE AND iContador_Pulsador=0 THEN
        iContador_Pulsador:=iContador_Pulsador+1;
    END_IF
    IF In_xMando=FALSE AND iContador_Pulsador=1 THEN
        Out_xConmutar1:=NOT Out_xConmutar1;
        iContador_Pulsador:=0;
    END_IF
```

Figura 3-5 - Código pulsador

Cuando se oprime el pulsador y la variable auxiliar está a 0, esta se cambia a 1, y en el momento que se deja de oprimir el pulsador, combinado con la variable auxiliar en valor 1, se puede alterar el estado de la salida.

- ***In_iSel_Modo=3:*** Modo varias salidas, se trata de una variación que aprovecha el comportamiento del pulsador. En el momento que el pulsador se ha activado y la variable auxiliar ha cambiado, esta última acciona 2 temporizadores, los cuales están programados para 0,5 y 1,2 segundos.

  Al dejar de oprimir el pulsador se detienen las temporizaciones, y dependiendo de su resultado se produce un comportamiento diferente.

```
ELSIF In_iSel_Modo=3 THEN

    Out_sMsgModo:='Pul. 2 salidas';

    IF In_xMando=TRUE AND iContador_Comport=0 THEN
        iContador_Comport:=iContador_Comport+1;
        xPulsado:=FALSE;
    END_IF

    TON_C1(IN:=iContador_Comport=1 , PT:=t_TPulsado , Q=>xComportamiento);
    TON_C2(IN:=iContador_Comport=1 , PT:=t_TPulsado2 , Q=>xComportamiento2);

    IF In_xMando=FALSE AND iContador_Comport=1 THEN
        xPulsado:=TRUE;
        iContador_Comport:=0;
    END_IF

    IF xPulsado=TRUE THEN

        IF NOT xComportamiento AND NOT xComportamiento2 THEN
            Out_xConmutar1:=NOT Out_xConmutar1;
        ELSIF xComportamiento AND NOT xComportamiento2 THEN
            Out_xConmutar2:=NOT Out_xConmutar2;
        ELSE
            IF NOT Out_xConmutar1 AND NOT Out_xConmutar2 THEN
                Out_xConmutar1:=Out_xConmutar2:=TRUE;
            ELSE
                Out_xConmutar1:=Out_xConmutar2:=FALSE;
            END_IF
        END_IF
        xPulsado:=FALSE;
    END_IF
```

Figura 3-6 - Código pulsador, varias salidas

Si no ha acabado ninguna de las temporizaciones, la pulsación ha durado menos de 0,5s y se actúa sobre la primera salida.

Si acaba la primera temporización, pero no la segunda, se actuará solo sobre la segunda salida.

Por último, si acaban las 2 temporizaciones, se actúa sobre las 2 salidas al mismo tiempo dependiendo del estado de las salidas en ese momento. Si alguna de las salidas esta activa, desactiva ambas, en caso contrario, activa ambas.

- ***In_iSel_Modo=4**:* Modo detector, este bloque utiliza la función TOF, la cual pone a FALSE la salida, al hacerse FALSE la entrada tras un tiempo determinado. Se hace necesaria a su vez una pequeña conversión, para transformar la entrada de usuario a tipo TIME, necesaria para hacer funcionar el TOF.

```
ELSIF In_iSel_Modo=4 THEN

    Out_sMsgModo:='Det. presencia';

    t_TApagado:= INT_TO_TIME(In_TApagado*1000);
    TOF_D(IN:=In_xMando , PT:=t_TApagado , ET=>tMostrar_TApagado, Q=>Out_xConmutar1);

    IF In_xMando=TRUE THEN
        Out_xConmutar1:=TRUE;
    END_IF
```

Figura 3-7 - Código detector de presencia

Cuando la entrada se activa también lo hace la salida, y en el momento que la entrada se desactiva, la función TOF se encarga de desactivar la salida tras el tiempo que se le haya marcado.

- ***In_iSel_Modo=5:*** Modo horario, esta parte del código compara las horas asignadas por el usuario en las entradas con la hora en ese momento, y distingue entre estar dentro del periodo horario en el que activar la salida y cuando no.

```
ELSIF In_iSel_Modo=5 THEN

    Out_sMsgModo:='Prog. Horaria';

    IF In_todHoraI<In_todReloj AND In_todReloj<In_todHoraF THEN
        Out_xConmutar1:=TRUE;
    ELSE
        Out_xConmutar1:=FALSE;
    END_IF
```

Figura 3-8 - Código horario luces

- ***In_iSel_Modo=6:*** Modo regulador, en este modo, siendo la entrada la intensidad luminosa deseada, primero se evalúa si la referencia ha cambiado respecto a un valor anterior, y como segundo paso si el mando esta activo o no.

  Si la intensidad deseada es mayor a cierto valor, se aplica una sencilla recta de conversión para trasladar el valor correcto al elemento de regulación, 0-100 a 0-30000, después de hacer este cambio se guarda el valor de la entrada, para repetir el ciclo nuevamente cuando se realice un cambio.

```
ELSIF In_iSel_Modo=6 THEN

    Out_sMsgModo:='Regulador';

    IF iReguladorAnt<>In_iIntensidadRegulador THEN
        xCambio:=TRUE;
    END_IF

    IF In_xMando AND xCambio THEN
        IF  (In_iIntensidadRegulador)>4 THEN
            Out_iLuminosidad:=(In_iIntensidadRegulador)*30000/100;
            Out_xConmutar1:=TRUE;
        ELSE
            Out_iLuminosidad:=0;
            Out_xConmutar1:=FALSE;
        END_IF
        iReguladorAnt:=In_iIntensidadRegulador;
        xCambio:=FALSE;
    ELSIF NOT In_xMando THEN
        Out_iLuminosidad:=0;
        Out_xConmutar1:=FALSE;
    END_IF
```

Figura 3-9 - Código regulador

- ***In_iSel_Modo=7:*** Modo regulador con sensor, difiere bastante en comportamiento al querer incluir un sensor de luminosidad respecto al modo anterior, haciéndose necesaria la inclusión de un regulador PID para gobernar la salida.

  Lo primero es una rutina con un TON y una serie de variables auxiliares para retrasar la actuación. Esto se hace para que al sistema le dé tiempo a alterar la salida y detectar el nuevo nivel de luminosidad antes de seguir intentando actuar sobre ella.

  De un modo similar se juzga si la temporización ha terminado y el estado del mando, para activar o no el PID que controla el valor de la salida.

  Al igual que el modo 6, se utiliza la misma recta de conversión para trasladar el valor al elemento de regulación.

```
ELSIF In_iSel_Modo=7 THEN

    Out_sMsgModo:='Regulador con sensor';

    TON_Estatico(IN:=xCambio,PT:=T#100MS,Q=>xEstatico);
    rActual:= TO_REAL(In_iSensorLuminosidad);
    rSet:= TO_REAL(In_iIntensidadRegulador);

    IF xEncendido=FALSE THEN
        xCambio:=TRUE;
        xEncendido:=TRUE;
    END_IF

    IF xEstatico AND In_xMando THEN
        Out_xConmutar1:=TRUE;
        xReset:=FALSE;
        xCambio:=FALSE;
        xEncendido:=FALSE;

    ELSIF NOT In_xMando THEN
        xReset:=TRUE;
        rSalidaRegulacion:=0;
        Out_xConmutar1:=FALSE;
    END_IF

    PID_REG (
    ACTUAL:= rActual ,
    SET_POINT:= rSet,
    KP:= In_rPID_KP ,
    TN:= In_rPID_TN,
    TV:= In_rPID_TV ,
    Y_MANUAL:= 0.0,
    Y_OFFSET:= ,
    Y_MIN:= 0.0,
    Y_MAX:= 100.0 ,
    MANUAL:= FALSE,
    RESET:=  xReset ,

    Y=> rSalidaRegulacion,
    LIMITS_ACTIVE=> ,
    OVERFLOW=> );

Out_iLuminosidad:=TO_INT(rSalidaRegulacion)*30000/100;

END_IF
```

Figura 3-10 - Código regulador con sensor

3.3.- Bloque funcional de climatización

Para el control de temperaturas en la instalación, se construye una función que permita establecer una temperatura objetivo, y según ciertos parámetros introducidos por el usuario, se pueda alcanzar y mantener esa temperatura. Algunos usos comerciales son la programación horaria y la generación de mensajes de error.

Características concretas desarrolladas para este bloque son:

- **Selector de modo de funcionamiento:** calefacción, aire acondicionado o climatizador en caso de que la instalación solo disponga de una de las funciones.
- **Selector de modo de activación:** elegir si activar la funcionalidad en modo manual o programación horaria.
- **Generación de mensaje de** error: en caso de no conseguir cambiar la temperatura en cierto tiempo, se avisa al usuario y se detiene la funcionalidad hasta que se reinicie.
- **Utilización de un PID:** para la regulación en los 3 modos de funcionamiento.

Si el bloque se activa en modo manual, intenta inmediatamente modificar la temperatura de la sala hasta igualarla con la temperatura objetivo, subiéndola en modo calefacción, bajándola en modo aire, o las dos acciones en modo clima. Si se activa en modo horario intenta realizar lo mismo, pero solo dentro de las horas que se le marquen.

En caso de no conseguir modificar la temperatura en al menos 0,1ºC al cabo de 30 segundos cuando debe estar activo, se considera como un error y se desactiva la funcionalidad. Esta comprobación se realiza siempre y cuando la diferencia entre las temperaturas sea mayor a 1ºC.

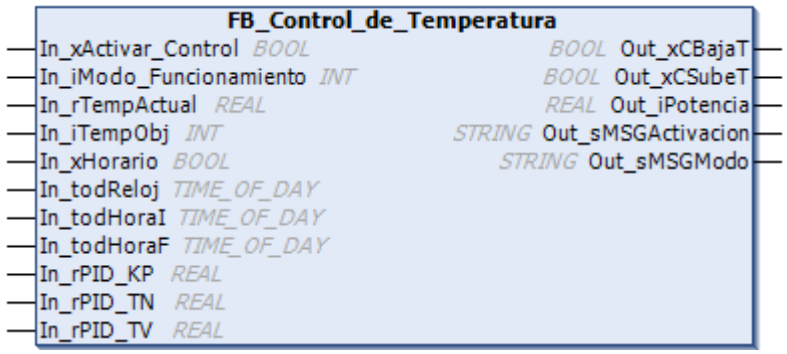

Figura 3-11 - Representación del bloque clima

3.3.1.- Entradas/Salidas

Variable	Descripción
In_xActivar_Control	Entrada para activar la funcionalidad
In_iModo_Funcionamiento	Entrada para elegir modo de control de temperatura
In_rTempActual	Entrada de sonda de temperatura
In_iTempObj	Entrada para indicar temperatura objetivo
In_xHorario	Entrada para poner en modo horario
In_todReloj	Entrada para la hora del reloj
In_todHoraI	Entrada para la hora de inicio
In_todHoraF	Entrada para la hora de finalización
In_rPID_KP	Entrada para configuración de acción proporcional
In_rPID_TN	Entrada para configuración de acción integral
In_rPID_TV	Entrada para configuración de acción derivativa
Out_xCbajaT	Salida correspondiente a la bajada de temperatura
Out_xCSubeT	Salida correspondiente a la subida de temperatura
Out_rPotencia	Salida para regular la potencia del equipo conectado
Out_sMSGActivacion	Salida para mostrar actividad
Out_sMSGModo	Salida para mostrar modo de uso

Tabla 3-2 - E/S del bloque clima

3.3.2.- Código

El código de este bloque funcional al ser más complejo está dividido en más partes. La primera parte correspondería a la activación, donde se distingue entre activar en modo de control horario o sin él, cambiando el mensaje de activación acorde a cada caso. La combinación de variables auxiliares *xIniciar* y *xEnEspera* que se definen en esta parte son una de las condiciones para hacer funcionar el bloque.

```
IF In_xActivar_Control AND NOT In_xHorario THEN
    xIniciar:=TRUE;
    Out_sMSGActivacion:='Manual activo';
    xEnEspera:=FALSE;
ELSIF In_xActivar_Control AND In_xHorario THEN
    IF In_todHoraI<In_todReloj AND In_todReloj<In_todHoraF THEN
        xIniciar:=TRUE;
        Out_sMSGActivacion:='Horario activo';
        xEnEspera:=FALSE;
    ELSE
        xIniciar:=FALSE;
        Out_sMSGActivacion:='Horario inactivo';
        xEnEspera:=TRUE;
    END_IF
END_IF
```

Figura 3-12 - Código de activación clima

El siguiente extracto de código sería la estructura para determinar si hay algún tipo de problema. Se ha programado para conseguir el siguiente efecto, una vez se dé la orden de activación a alguna de las salidas, si la temperatura ambiente no ha variado al menos en 0,1ºC al cabo de 1 minuto, se tomará como error y se desactivara la funcionalidad. También se ha tenido en cuenta que esta comprobación solo se haga fuera de la zona “estable” de funcionamiento, la cual se ha definido como +-1ºC respecto a la temperatura objetivo que se haya marcado.

```
Inicio(CLK:= xIniciar);
    IF Inicio.Q THEN
        rTempError:=In_rTempActual;
    END_IF
Reinicio(CLK:= xEstable);
    IF Reinicio.Q THEN
        rTempError:=In_rTempActual;
    END_IF

TON_Error(In:=xTempError,PT:=T#30S,Q=>xComprobar);

IF NOT xTempError AND NOT xEnEspera THEN
    xTempError:=TRUE;
END_IF

Comprobar(CLK:= xComprobar);
IF Comprobar.Q THEN
    IF ABS(In_rTempActual - rTempError)<0.1 AND NOT xEstable THEN
        xError:= TRUE;
    ELSE
        xTempError:=FALSE;
        rTempError:=In_rTempActual;
    END_IF
END_IF

IF xError THEN
    Out_sMSGActivacion:='Error';
    xIniciar:=FALSE;
    Out_xCSubeT:=FALSE;
    Out_xCBajaT:=FALSE;
    xEstable:=FALSE;
    xEnEspera:=TRUE;
END_IF
```

Figura 3-13 - Código rutina de error para el clima

Una vez comprobada la activación y el error, también se integra el selector de modo, para funcionar en uno de los modos 3 definidos, calefacción, aire, o climatizador. El extracto de la figura siguiente corresponde al modo calefacción.

```
IF iModo_Fun=1 AND NOT xError THEN

    IF (In_iTempObj - In_rTempActual)<=1.1 THEN
            xEstable:=TRUE;
        ELSE
            xEstable:=FALSE;
        END_IF

    IF xIniciar THEN
        IF xCalentar=TRUE THEN
            IF (In_rTempActual - 1)<In_iTempObj THEN
                Out_xCSubeT:=TRUE;
            ELSE
                xCalentar:=FALSE;
                Out_xCSubeT:=FALSE;
            END_IF
        END_IF
        IF xCalentar=FALSE THEN
            IF (In_rTempActual + 1)>In_iTempObj THEN
                Out_xCSubeT:=FALSE;
            ELSE
                xCalentar:=TRUE;
            END_IF
        END_IF
    ELSE
        Out_xCSubeT:=FALSE;
        xError:= FALSE;
        xTempError:=FALSE;
    END_IF

END_IF
```

Figura 3-14 - Código calefacción

El código comprueba las condiciones de modo y error, a fin de diferenciar en qué estado debe entrar, calefacción, aire acondicionado, climatizador o error.

Una vez comprobado, el primer tramo de código tiene la función de cambiar el valor de la variable *xEstable*, utilizada para la rutina de comprobación de error de la temperatura. Lo hace comparando la temperatura objetivo y la que recibe de la sonda de temperatura con condiciones específicas para cada modo, pero básicamente el resultado es estar dentro o fuera de la zona "estable" de funcionamiento.

Si las condiciones anteriores se han cumplido, en la siguiente parte de código se comprueba la variable *xIniciar* que marca el inicio de la funcionalidad. A continuación, se evalúa la

variable *xCalentar,* con esta variable el programa distingue cual fue la última acción que tomo, si calentó la última vez o estaba apagado. Después compara las entradas de temperaturas, y dependiendo del resultado, activará la salida, o la detendrá cuando se acerque a un límite establecido. El bloque de aire funciona exactamente igual con sus propias variables, su salida para enfriar y los sentidos de las comparaciones de temperaturas invertidos como cabe esperar.

El bloque de climatización es más extenso, pues combina las funciones de los 2 anteriores. El primer paso que da después de comprobar las activaciones es saber si esta fuera de la zona estable, que se ha definido como +- 1ºC respecto a la temperatura marcada. En caso de estar fuera de esta zona, se comprueban si se está por encima o por debajo de ella, y en función de la solución se procederá a calentar o enfriar, utilizando las mismas variables auxiliares de los modos anteriores. De este modo el bloque se mantendrá realizando una sola de las acciones, calentar o enfriar, hasta que se salga de esta zona estable y deba comprobar que hacer de nuevo.

Por último y para todos los modos, cuando se sabe que acción realizar todavía es necesario ofrecer un valor de potencia en la salida. Para ello se utiliza un control por PID, al que se le invierten las entradas en función de si debe enfriar o calentar, para siempre ofrecer una potencia positiva.

```
rTempObj:= TO_REAL(In_iTempObj);

IF Out_xCSubeT THEN
    rActual:=In_rTempActual;
    rSet:=rTempObj;
    xReset:=FALSE;
ELSIF Out_xCBajaT THEN
    rActual:=rTempObj;
    rSet:=In_rTempActual;
    xReset:=FALSE;

ELSIF NOT Out_xCBajaT AND NOT Out_xCSubeT THEN
    Out_iPotencia:=0;
    xReset:=TRUE;
END_IF

    PID_TEMP (
    ACTUAL:= rActual ,
    SET_POINT:= rSet,
    KP:= In_rPID_KP ,
    TN:= In_rPID_TN,
    TV:= In_rPID_TV ,
    Y_MANUAL:= 0.0,
    Y_OFFSET:= ,
    Y_MIN:= 0.0,
    Y_MAX:= 100.0 ,
    MANUAL:= FALSE,
    RESET:=  xReset ,

    Y=> Out_iPotencia,
    LIMITS_ACTIVE=> ,
    OVERFLOW=> );
```

Figura 3-15 - Regulación de potencia clima

3.4.- Bloque funcional de alarmas

Para la seguridad de los usuarios se añaden funciones para la gestión de alarmas, tanto técnicas como de intrusión. Algunos usos comerciales para intrusión son la activación/desactivación por contraseña y la distinción entre intruso y propietario. Para las técnicas el cortar de inmediato el agua o gas en función del problema que se presente.

Debido a las claras diferencias de comportamiento que presentan las alarmas de intrusión y técnicas, se considera que son lo suficientemente distintas como para separarlas en bloques funcionales diferentes, los cuales se ven a continuación.

3.4.1.- Alarma de intrusión

A este bloque se le han añadido funciones concretas como:

- **Generar mensajes:** estos irán acordes al estado actual.
- **Propietario vs Intruso:** Capacidad para distinguir entre sensores de intrusión y propietario.
- **Contraseña:** incluir la activación mediante una y además permitir su cambio.

El bloque funcional seguirá el siguiente comportamiento, se activará solo cuando la entrada de intrusión esta desactivada y no se esté en el modo configuración, en cuyo caso pasará a pedir la contraseña. En caso de introducirla correctamente se iniciará un periodo de activación antes de que quede completamente activada. Después de estar completamente activada pueden pasar 2 cosas, si se activa la entrada de intrusión se activa la alarma directamente, si se activa la entrada de propietario se dispone de un tiempo para desactivar la función.

El método para detener la alarma estando activa o cuando ya ha saltado es introducir la contraseña correctamente lo que permitirá que el botón de desactivación funcione, en caso contrario, aunque se presione el botón de desactivación no surtirá ningún efecto.

Al modo configuración solo se puede acceder si la alarma está completamente desactivada. En este modo se puede cambiar la contraseña, introduciéndola correctamente, para a continuación introducir la nueva contraseña 2 veces, la segunda como confirmación.

En todo momento los mensajes generados informaran de la introducción incorrecta de la contraseña, mal funcionamiento de sensores, activación de la alarma o inactividad.

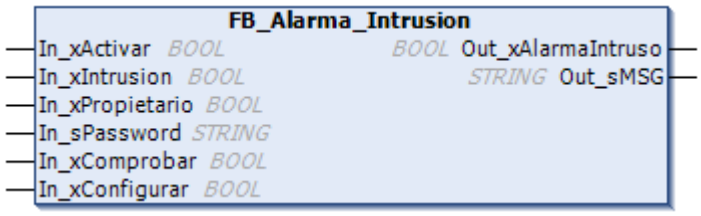

Figura 3-16 - Representación del bloque de alarmas de intrusión

3.4.1.1.- Entradas/Salidas

Variable	Descripción
In_xActivar	Entrada para activar/desactivar la funcionalidad
In_xIntrusion	Entrada para sensores considerados como intrusión directa
In_xPropietario	Entrada para sensores de entrada del propietario
In_sPassword	Entrada para escribir la contraseña
In_xComprobar	Entrada para validar/comprobar la contraseña
In_xConfigurar	Entrada para acceder a modo configurar
Out_xAlarmaIntruso	Salida para indicar que ha saltado aviso de alarma
Out_sMSG	Salida para mensajes de estado

Tabla 3-3 - E/S del bloque intrusión

3.4.1.2.- Código

El primer paso es comprobar la activación, que depende de accionar la entrada de activación, no estar en modo configurar y no estar activa la entrada de intrusión. Si todo se cumple se da paso a la espera de la introducción de la contraseña.

Al oprimir el botón de comprobación, se comparará la contraseña introducida con la guardada, la cual es por defecto “1234”, y se generaran mensajes acordes con el resultado.

```
IF In_xActivar AND iEstado=0 AND NOT In_xIntrusion AND NOT In_xConfigurar THEN

    sPasswordConfig:=' ';
    iConfig:=0;
    Out_sMSG:='Esperando contraseña';
    xEsUsuario:=TRUE;

    IF In_xComprobar THEN
        sPasswordUsu:=In_sPassword;
        IF  sPasswordUsu=sPassword THEN
                xEsUsuario:=FALSE;
                xArme:=TRUE;
                iEstado:=1;
        ELSE
                Out_sMSG:='Contraseña incorrecta';
        END_IF
    END_IF

ELSIF NOT In_xActivar AND NOT In_xIntrusion AND NOT In_xConfigurar THEN
    Out_sMSG:='Alarma desactivada';
    sPasswordConfig:=' ';
    iConfig:=0;

ELSIF In_xIntrusion THEN
    Out_sMSG:='Problema en los sensores';
END_IF
```

Figura 3-17 - Código iniciación de intrusión

En caso de ser correcta se pone en marcha la temporización para ofrecer tiempo al usuario de salir antes de la activación completa, esto corresponde al estado 1 de la alarma.

```
IF iEstado=1 THEN
    Out_sMSG:='Alarma activandose';
    IF xActivar THEN
        sPasswordUsu:=' ';
        iEstado:=2;
        xArme:=FALSE;
    END_IF
END_IF
```

Figura 3-18 - Código espera de activación

Cuando esta temporización termina se pasa al estado 2, en el que la funcionalidad se encuentra a la espera de que se active alguno de las entradas que corresponden a los sensores.

```
IF iEstado=2 THEN

    Out_sMSG:=('Alarma activa');

    IF In_xPropietario THEN
        iEstado:=3;
        xTdeGracia:=TRUE;
    ELSIF In_xIntrusion THEN
        iEstado:=4;
        Out_xAlarmaIntruso:=TRUE;
        END_IF
END_IF
```

Figura 3-19 - Código alarma activa

En caso de activarse la entrada de propietario se activa otra temporización para ofrecer un periodo de tiempo en el que el usuario pueda desactivar la alarma y se pasa al estado 3.

```
IF iEstado=3 THEN
    IF NOT xEsUsuario THEN
        Out_sMSG:=('Esperando contraseña');
    ELSE
        Out_sMSG:=('Secuencia detenida');
    END_IF
    IF In_xComprobar THEN
        sPasswordUsu:=In_sPassword;
        IF  sPasswordUsu=sPassword THEN
                sPasswordUsu:=' ';
                xEsUsuario:=TRUE;
                xTdeGracia:=FALSE;
        END_IF
    END_IF

    IF xIntruso THEN
        iEstado:=4;
        Out_xAlarmaIntruso:=TRUE;
    ELSIF In_xIntrusion THEN
        iEstado:=4;
        Out_xAlarmaIntruso:=TRUE;
    END_IF
END_IF
```

Figura 3-20 - Código entrada de propietario

Con una estructura similar a la parte de código para la activación por contraseña, esta parte del código aprovecha la variable auxiliar *xEsUsuario* para acceder a una parte del programa que desactiva completamente la alarma si se consigue introducir la contraseña

correctamente. Si no se introduce la contraseña correcta antes de que acabe el tiempo, o se activa la entrada de intrusión, salta la alarma y se pasa al estado 4.

```
IF iEstado=4 THEN

    xTdeGracia:=FALSE;

    IF Out_xAlarmaIntruso THEN
        Out_sMSG:='Intruso detectado';
    END_IF

    IF In_xComprobar THEN
        sPasswordUsu:=In_sPassword;
        IF  sPasswordUsu=sPassword THEN
                sPasswordUsu:=' ';
                xEsUsuario:=TRUE;
                xTdeGracia:=FALSE;
        END_IF
    END_IF

END_IF
```

Figura 3-21 - Código intrusión detectada

En este estado se está esperando la correcta introducción de la contraseña para desactivar la alarma, en caso afirmativo también se utiliza la variable *xEsUsuario* para acceder a la parte del código que desactiva la funcionalidad.

```
IF NOT In_xActivar AND xEsUsuario THEN
    iEstado:=0;
    xArme:=FALSE;
    Out_sMSG:='Alarma desactivada';
    xEsUsuario:=FALSE;
    Out_xAlarmaIntruso:=FALSE;
END_IF
```

Figura 3-22 - Código desactivación

Una vez la variable *xEsUsuario*, en combinación con la entrada de activación se puede finalmente desactivar la función.

3.4.2.- Alarmas técnicas

Funciones concretas para este bloque son:

- **Válvulas:** permitir su manejo manual y cierre automático en caso de alarma.
- **Estado de alarma:** impedir el uso de válvulas, o desconexión de la funcionalidad en caso de alarma.

Este bloque permite el control manual de las válvulas de agua y gas. El comportamiento de alarmas se activa o desactiva de forma independiente a ese manejo de válvulas, y en caso de estar activo y producirse una alarma, arrebata el control de dichas válvulas al usuario y las mantiene desactivadas hasta que la alarma haya desaparecido.

En caso de producirse una alarma se generan mensajes en función del tipo de alarma que se produzca. Además, no se permite la desactivación del comportamiento de la alarma si las entradas de los sensores se encuentran activas, pero si se puede utilizar la entrada de acuse para detener momentáneamente el aviso mientras se subsana la avería. Esta misma entrada de acuse reinicia la alama cuando ya no existe ningún problema.

Figura 3-23 - Representación del bloque de alarmas técnicas

3.4.2.1.- Entradas/Salidas

Variable	Descripción
In_xActivar	Entrada para activar la parte de gestión de alarmas
In_xAcuse	Entrada para detener el aviso o reiniciar las alarmas
In_xDetFuego	Entrada para detector de incendios
In_xDetGas	Entrada para detector de gas
In_xDetInun	Entrada para detector de inundación
In_xValvGas	Entrada para el manejo de la válvula de gas
In_xValvAgua	Entrada para el manejo de la válvula de agua
Out_xAlarmaTecnica	Salida para aviso de alarmas
Out_xValvGas	Salida para el manejo de la válvula de gas
Out_xValvAgua	Salida para el manejo de la válvula de agua
Out_sMSG	Salida para mostrar mensajes de aviso

Tabla 3-4 - E/S bloque técnicas

3.4.2.2.- Código

Lo primero que hace este bloque es distinguir cuando dar o no la capacidad de modificar el estado de las válvulas al usuario, de modo que si alguna de las variables auxiliares *xProblema* esta activa, las cuales solo cambian de estado cuando se da algún caso de alarma, se impide la modificación.

```
IF NOT xProblemaGas AND NOT xProblemaFuego THEN
    Out_xValvGas:=In_xValvGas;
END_IF

IF NOT xProblemaAgua THEN
    Out_xValvAgua:=In_xValvAgua;
END_IF
```

Figura 3-24 - Código válvulas

A continuación, se evalúa la activación y la desactivación de la parte de alarmas, con la condición específica de que no se pueda desactivar si existe algún problema en ese momento, de esta manera se evita que se pueda apagar la funcionalidad en mitad de un estado de alarma.

```
IF In_xActivar THEN
    xActivacion:=TRUE;
ELSIF NOT In_xActivar AND NOT xProblemaGas AND NOT xProblemaAgua AND NOT xProblemaFuego THEN
    xActivacion:=FALSE;
    Out_sMSG:='Alarma desconectada';
END_IF
```

Figura 3-25 - Código de activación técnicas

Una vez que ha activado la funcionalidad, se evaluara el estado de las entradas de los detectores, utilizando las variables auxiliares *xProblema*. La motivación del uso de estas variables es que el estado de alarma permanezca, aunque pueda estropearse el detector asociado por alguna razón, y así no permitir retomar el control de las válvulas espontáneamente. Al activarse un sensor, su variable auxiliar cambia, se da la alarma y se manda un mensaje con el tipo de problema encontrado.

```
IF xActivacion THEN

    IF NOT xProblemaGas AND NOT xProblemaAgua AND NOT xProblemaFuego THEN
        Out_sMSG:='Alarma conectada';
    END_IF

    IF In_xDetGas AND NOT xProblemaGas THEN
        Out_xValvGas:=FALSE;
        Out_sMSG:='Escape de gas detectado';
        Out_xAlarmaTecnica:=TRUE;
        xProblemaGas:=TRUE;
    END_IF
```

Figura 3-26 – Código de evaluación de detectores

Una vez la alarma se haya activado, entra en juego la entrada de acuse, que podrá tener 2 efectos diferentes.

El primer efecto es el de apagar la secuencia de alarma cuando se produce, para poder subsanar el problema una vez detectado de una tranquila, sin embargo, hacer esto no impide que si se produce otro problema distinto no se vuelva a activar la secuencia. En este estado sigue impidiendo el manejo de las válvulas. El segundo caso es que no haya ningún detector activo, el efecto que tiene es el mismo que reiniciar la alarma al estado inicial.

```
IF In_xAcuse AND Out_xAlarmaTecnica THEN
        Out_xAlarmaTecnica:=FALSE;
        Out_sMSG:='Aviso Detenido';
END_IF

IF In_xAcuse AND NOT In_xDetGas THEN
        Out_xAlarmaTecnica:=FALSE;
        xProblemaGas:=FALSE;
END_IF
```

Figura 3-27 - Código de acuse de alarmas técnicas

3.4.3.- Configuración

Este último bloque de alarmas surge como nexo de unión para los 2 anteriores en caso de querer utilizar los mismos elementos de aviso para ambas funciones. Además, permite cierta modificación de la secuencia luminosa/sonora que se produce cuando se entra en estado de alarma e impide cambios en la secuencia si hay alguna alarma activa en ese momento.

Permite activar/desactivar cada elemento, y cambiar la frecuencia de funcionamiento entre 0,5 y 1 segundo.

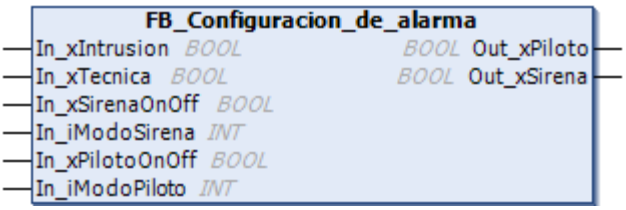

Figura 3-28 - Representación del bloque configuración de alarma

3.4.3.1.- Entradas/Salidas

Variable	Descripción
In_xIntrusion	Entrada para aviso intrusión
In_xTecnica	Entrada para aviso técnico
In_xSirenaOnOff	Entrada de configuración para activar desactivar la sirena
In_iModoSirena	Entrada de configuración para elegir el comportamiento de la sirena
In_xPilotoOnOff	Entrada de configuración para activar desactivar el piloto
In_iModoPiloto	Entrada de configuración para elegir el comportamiento del piloto
Out_xPiloto	Salida para piloto luminoso
Out_xSirena	Salida para sirena

Tabla 3-5 - E/S bloque configuración

3.4.3.2.- Código

El código está pensado para funcionar mediante los avisos que generan los 2 bloques de alarmas anteriores, por lo tanto, si alguno de los 2 se activa se inicia la secuencia.

```
IF In_xIntrusion OR In_xTecnica THEN
    xActAlarma:=TRUE;
END_IF
```

Figura 3-29 - Código activación de secuencia

Si la secuencia de alarmas debe producirse, se entra en la siguiente parte del programa, donde se utilizan bloques BLINK. Estos bloques de función cambian el estado de las salidas a intervalos modificables tanto para el encendido y apagado. Aprovechando dos bloques, y con algo más de programación se consigue generar comportamientos diferentes para 2 salidas, pensadas para una sirena y un piloto luminoso, según la configuración elegida.

```
IF xActAlarma THEN

    BLINK_SIRENA(Enable:=xActAlarma , Timelow:=tOFFS , Timehigh:=tONS , Out=>xActivoS);
    BLINK_PILOTO(Enable:=xActAlarma , Timelow:=tOFFP , Timehigh:=tONP , Out=>xActivoP);

    IF xSirenaOnOff THEN
        IF  xSirenaCont THEN
            Out_xSirena:=TRUE;
        ELSE
            Out_xSirena:=xActivoS;
        END_IF
    END_IF

    IF xPilotoOnOff THEN
        IF xPilotoCont THEN
            Out_xPiloto:=TRUE;
        ELSE
            Out_xPiloto:=xActivoP;
        END_IF
    END_IF

END_IF
```

Figura 3-30 - Código de secuencia de alarmas

La configuración de dichos parámetros se encuentra en la siguiente parte del código, a la cual solo se accede cuando no se está produciendo ninguna alarma. Según los valores de las demás entradas del bloque, se puede elegir si utilizar el elemento de forma continua, con un periodo de 500ms, 1s, o no utilizarlo. Además, ambos elementos se pueden configurar por separado.

```
IF NOT In_xIntrusion AND NOT In_xTecnica THEN
    xActAlarma:=FALSE;
    Out_xPiloto:=FALSE;
    Out_xSirena:=FALSE;

    IF In_xSirenaOnOff THEN
        xSirenaOnOff:=TRUE;
        IF In_iModoSirena=0 THEN
            xSirenaCont:=TRUE;
        ELSIF In_iModoSirena=1 THEN
            xSirenaCont:=FALSE;
            tONS:= T#500MS;
            tOFFS:= T#500MS;
        ELSIF In_iModoSirena=2 THEN
            xSirenaCont:=FALSE;
            tONS:= T#1000MS;
            tOFFS:= T#1000MS;
        END_IF
    ELSE
        xSirenaOnOff:=FALSE;
    END_IF
```

Figura 3-31 - Código de configuración de secuencia

3.5.- Bloque funcional de riego

Otra de las funciones es la gestión de sistemas de riego, poder activarlos durante un tiempo bajo ciertas condiciones. Usos comerciales encontrados pueden ser, limitación por humedad del suelo y programación horaria. Algunas funciones incluidas en este bloque son:

- **Modos de activación:** automático/manual.
- **Configuración:** es posible alterar el tiempo de riego.
- **Sensor de humedad:** es posible alterar el límite de humedad en el suelo a partir de la cual no se activa el riego.
- **Sensor de lluvia:** a partir de cierto tiempo que se mantenga activo no permite el riego.

La funcionalidad permite mantener la salida hacia los aspersores o sistema de riego activa durante un tiempo determinado cuando se han cumplido las condiciones, y al pasar ese tiempo que se desactive automáticamente.

En modo manual la única condición es haberlo activado, no tendrá en cuenta humedad, la hora o lluvia, únicamente que el usuario haya elegido activar la función en ese momento.

Para el modo automático, se tienen en cuenta la activación, la humedad, la hora, y que la entrada del sensor de lluvia no haya estado más de cierto tiempo activa. Al cumplirse todas las condiciones se activa la función.

Figura 3-32 - Representación del bloque riego

3.5.1.- Entradas/Salidas

Variable	Descripción
In_xActivar_Bloque	Entrada para activar el bloque
In_xIniciar	Entrada para empezar la temporización
In_xModo	Entrada para elegir entre modo manual o automático/horario
In_xSensorLluvia	Entrada para el sensor de lluvia
In_iSensorHumedad	Entrada para colocar un sensor de humedad del terreno
In_iHumedadUsuario	Entrada para cambiar la humedad de activación por defecto 50%
In_todHora_Reloj	Entrada para la hora del reloj
In_todHora_Act	Entrada para la hora del usuario
In_TRiego	Entrada para definir el tiempo de riego
Out_tMostrarTRiego	Salida con la que poder visualizar el tiempo restante de riego
Out_xAspersor	Salida que actuara sobre los aspersores

Tabla 3-6 - E/S bloque riego

3.5.2.- Código

El código para este bloque funcional, a diferencia de los demás, se ha desarrollado en lenguaje de contactos, por añadir un bloque en otro lenguaje distinto demostrando esta capacidad del estándar IEC 61131-3.

Como en los demás bloques, la primera parte del código es la activación, donde se evalúan las entradas de activación del bloque, el modo de funcionamiento y el inicio de la funcionalidad.

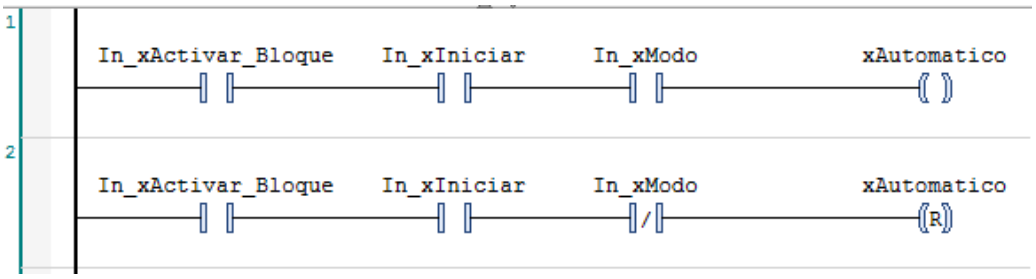

Figura 3-33 - Código de activación de riego

Con el código anterior se decide el iniciar la función en modo automático o por el contrario en manual. Si se hace en automático, las siguientes partes del código evalúan las condiciones necesarias para decidir si debe accionar la salida a los aspersores o no.

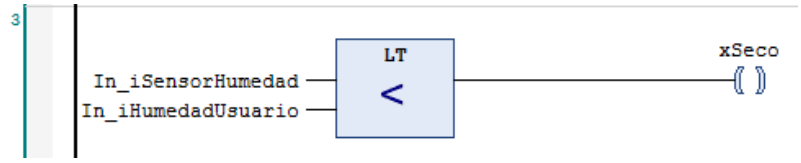

Figura 3-34 - Código humedad del suelo

La primera condición es la humedad en el suelo, comparar los 2 valores recibidos por las entradas modifica una variable auxiliar que indica el resultado de esa comparación.

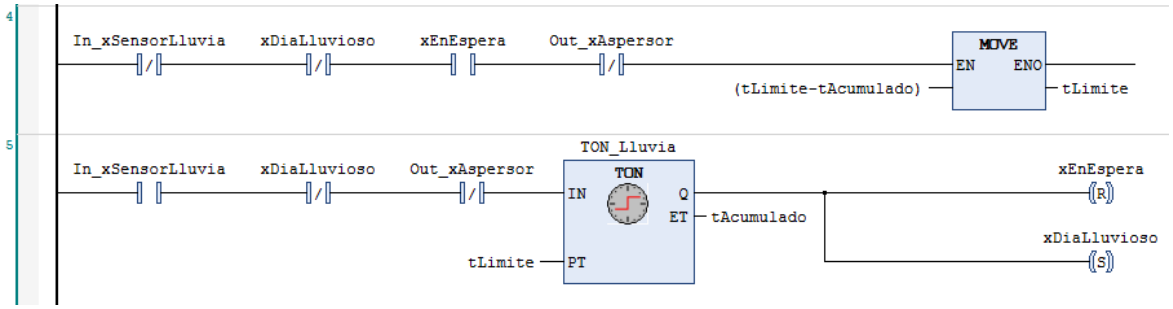

Figura 3-35 - Código acumulación de lluvia

La siguiente condición es juzgar si el día ha sido lluvioso, para ello, mientras la funcionalidad se encuentre a la espera de activarse se ira contando el tiempo que la entrada de lluvia esté activa a lo largo del día. En caso de superarse el límite de tiempo encontrándose en espera de activación, se impedirá que ese día se pueda activar la funcionalidad a través de la variable auxiliar *xDiaLluvioso*.

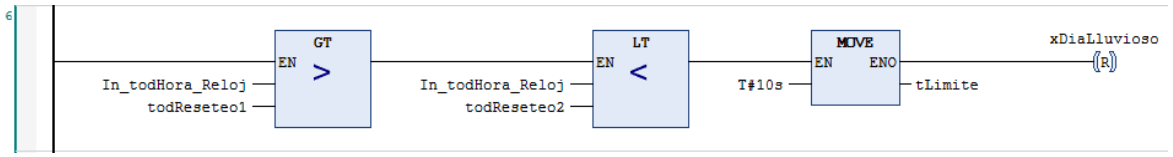

Figura 3-36 - Código de reinicio de lluvia

Cada día, entre las 00:00:00 y las 00:00:02, se reiniciarán las variables relacionadas con la acumulación de lluvia, el tiempo límite para la acumulación vuelve a su valor original, y se desactiva la variable *xDiaLluvioso*.

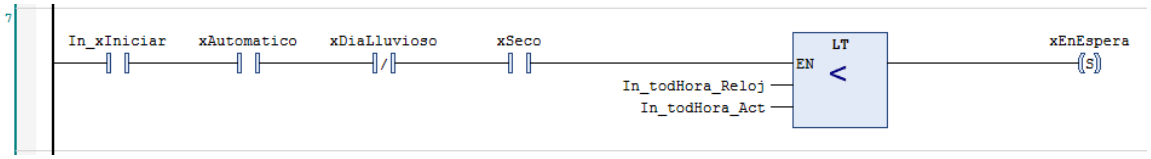

Figura 3-37 - Código condiciones de espera

El siguiente tramo de código tiene en cuenta las condiciones anteriores, además de comparar la hora de activación con la hora actual. Si se cumplen todas incluido que se está en una hora más temprana que la de activación estipulada, la funcionalidad entrara en espera de activación.

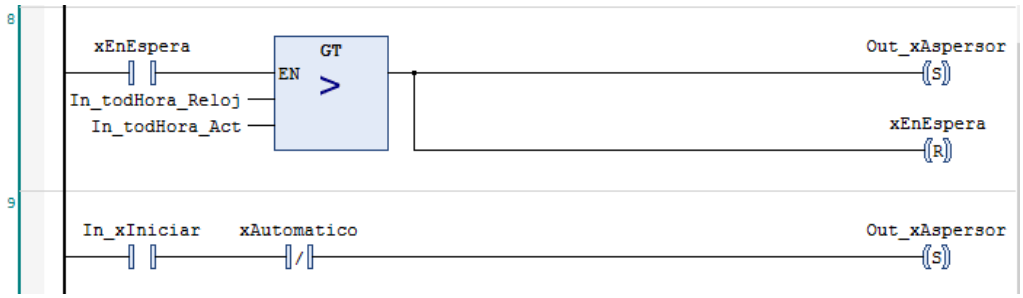

Figura 3-38 - Código de activación de riego

Una vez hechas todas las comprobaciones llega el momento de activar el riego. En modo automático mientras todas las condiciones anteriores se cumplan se está en modo espera, hasta que la hora de reloj sobrepase a la hora de activación del usuario, en ese momento se activa el riego. El modo manual en cambio no debe hacer ningún tipo de comprobación, y ese se activa directamente cuando se quiere iniciar la funcionalidad.

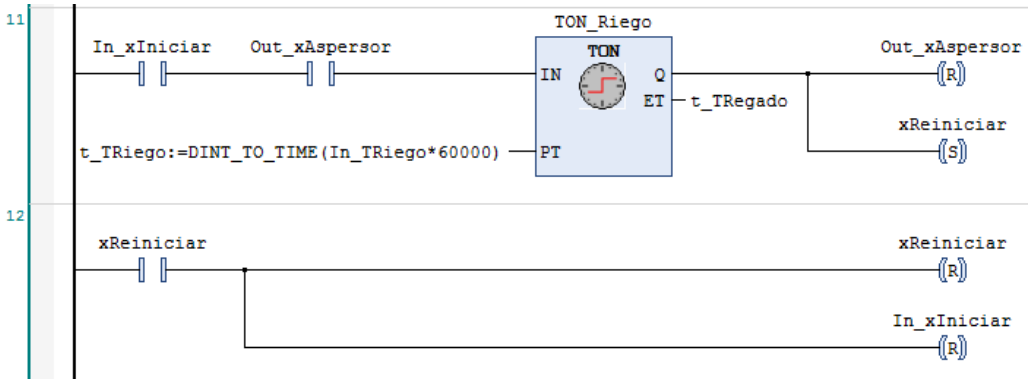

Figura 3-39 - Código riego

Una vez se ha iniciado el riego, esta parte del código se encarga de mantener activa la salida por el tiempo que haya introducido el usuario mediante una función TON, una funcionalidad que activa la salida durante un tiempo determinado cuando su entrada se activa. Una vez terminado se reinicia la funcionalidad a la espera del siguiente día, o activación manual.

3.6.- Bloque funcional de control de motores

Para el manejo de sistemas motorizados de persianas, toldos y lamas, se crea un bloque configurable con varias opciones para su funcionamiento. Algún ejemplo de uso comercial seria la reacción a la luz, o la detención por temporizador.

Algunas funciones incluidas en este bloque son:

- **Modos de activación:** manual a través de dos entradas o automático a través de sensor.
- **Alternativas de parada:** parada por medio de desactivación de entrada, tiempo o final de carrera.
- **Configuración**: se puede alterar el de tiempo de detención automática.

El funcionamiento del bloque es simple, se conecta en modo manual o automático y permanece en el elegido hasta la desconexión de toda la funcionalidad, o lo que es lo mismo no se puede cambiar el modo cuando está en funcionamiento.

En modo manual, a través de 2 entradas se puede ejercer control sobre el sistema, subiendo/abriendo y bajando/cerrando, acción que permanecerá activa hasta que se deje de oprimir el botón de orden, pase un tiempo modificable por el usuario o, en caso de disponer de ellos, se alcance el final de carrera correspondiente.

Si se activa en modo automático, a través del estado del detector, se ejerce una de las acciones correspondientes, que se detienen al cabo del tiempo estipulado o por el final de carrera.

La funcionalidad además está diseñada para que las 2 acciones no se puedan ejercer a la vez como precaución, y si se desconecta la funcionalidad en medio de una acción, esta termine antes de permitir hacer cualquier otra cosa.

Figura 3-40 - Representación del bloque motores

3.6.1.- Entradas/Salidas

Variable	Descripción
In_xActivar	Entrada para activar la funcionalidad
In_xAutomatico	Entrada para modo automático por detector de sol
In_xDet_Sol	Entrada para detector solar o luminosidad
In_xBajar	Entrada para desplazar en dirección descendente
In_xSubir	Entrada para desplazar en dirección ascendente
In_xFCI	Entrada para el final de carrera inferior
In_xFCS	Entrada para el final de carrera superior
In_iTiempoSubir	Tiempo especificado que la acción de subida estará activa, por defecto 5 segundos
In_iTiempoBajar	Tiempo especificado que la acción de bajada estará activa, por defecto 5 segundos
Out_xMotorB	Salida para indicar al motor que debe bajar/cerrar
Out_xMotorS	Salida para indicar al motor que debe subir/abrir
Out_MSGEstado	Salida para visualizar el mensaje de estado inactivo/manual o automático

Tabla 3-7 E/S bloque motores

3.6.2.- Código

El código para este bloque se ha realizado en SFC (*Sequential Function Chart*), al igual que el bloque de riego para demostrar que pueden funcionar.

Como el resto de los bloques el primer paso es distinguir entre los estados de activación, bien el bloque se encuentra desactivado, en modo manual o automático.

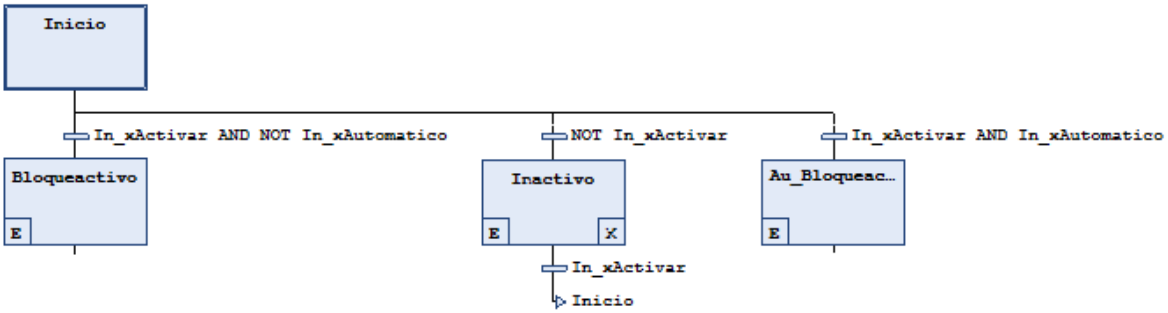

Figura 3-41 - Código selección de modo

En esta primera parte del código se decide avanzar por una de las 3 vertientes, en función de la combinación de las entradas de activación y automático. La funcionalidad está preparada para que no pueda saltar de una vertiente a otra a menos que la entrada *In_Activar* cambie de estado, lo que se traduce en que, si se quiere cambiar el modo de funcionamiento del bloque, este debe reiniciarse, de este modo se evitan problemas.

En caso de entrar en modo manual la funcionalidad sigue el código de la figura siguiente, en el que permanece hasta la desactivación. Ya que se trata de un esquema bastante grande se muestra únicamente la lógica para cuando se da la acción de bajar, ya que el subir sigue una mecánica similar con sus propias variables para conseguir el efecto contrario.

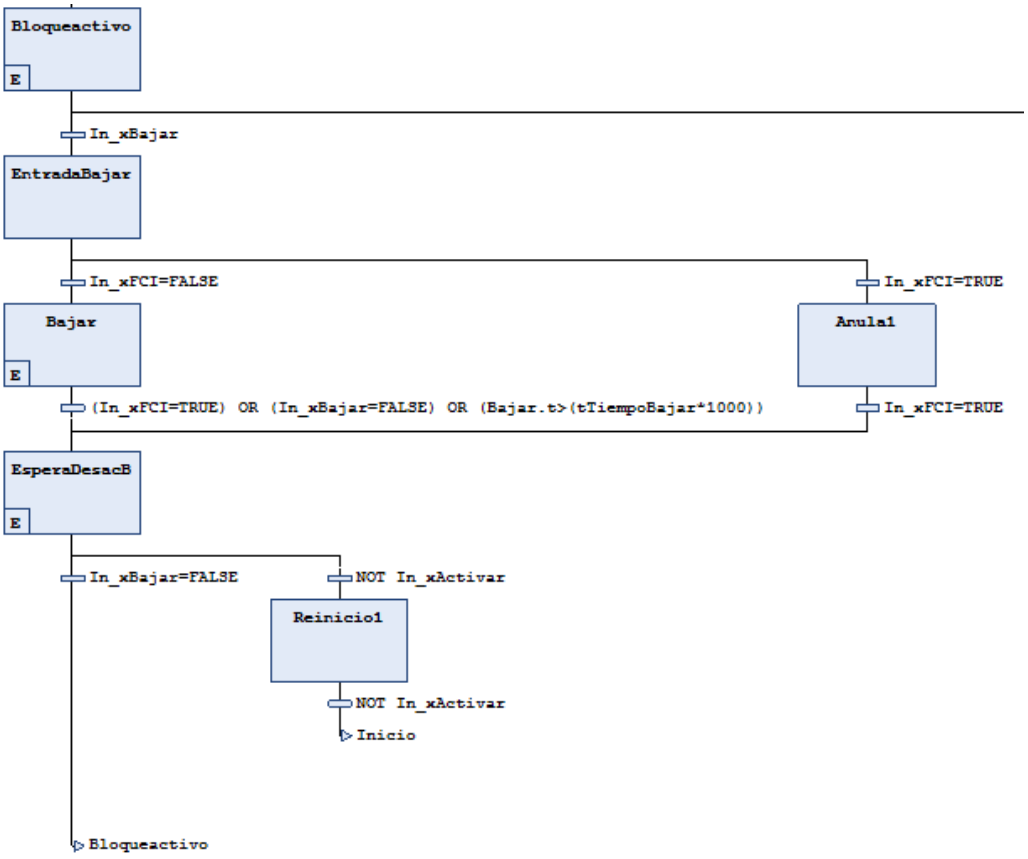

Figura 3-42 - Código activación manual

El bloque se encuentra a la espera en la fase *BloqueActivo* hasta que la entrada *In_xBajar* en este caso o *In_xSubir* en el caso contrario se accionen. Después pasa por la fase *EntradaBajar* que se encuentra aquí únicamente porque este lenguaje no permite hacer 2 evaluaciones de variables seguidas sin que exista una fase entre ellas.

El siguiente paso es evaluar finales de carrera, si se han añadido se podrá diferenciar el caso en el que inferior esté activo, indicando que el sistema motorizado no puede seguir bajando de modo que se anula la orden. Si no hay finales de carrera, su valor siempre por defecto siempre es FALSE, por lo que se ejecuta la acción de la fase *Bajar*, que activa la salida *Out_xMotorB.*

Esta acción se mantiene activa hasta que se cumple alguna de las condiciones de salida, bien se active el final de carrera correspondiente, deje de haber acción de la entrada *In_xBajar*, o pase un tiempo determinado que puede alterar el usuario. Después de haber acabado esta

fase, se evalúa que la acción de entrada se haya desactivado antes de volver a la fase de espera, evitando así que se puedan activar las 2 entradas a la vez o que se ejecute en bucle.

Por último, existe el modo automático, que funciona a través de una sola entrada booleana, ejecutando acciones para subir y bajar de una manera similar al modo manual.

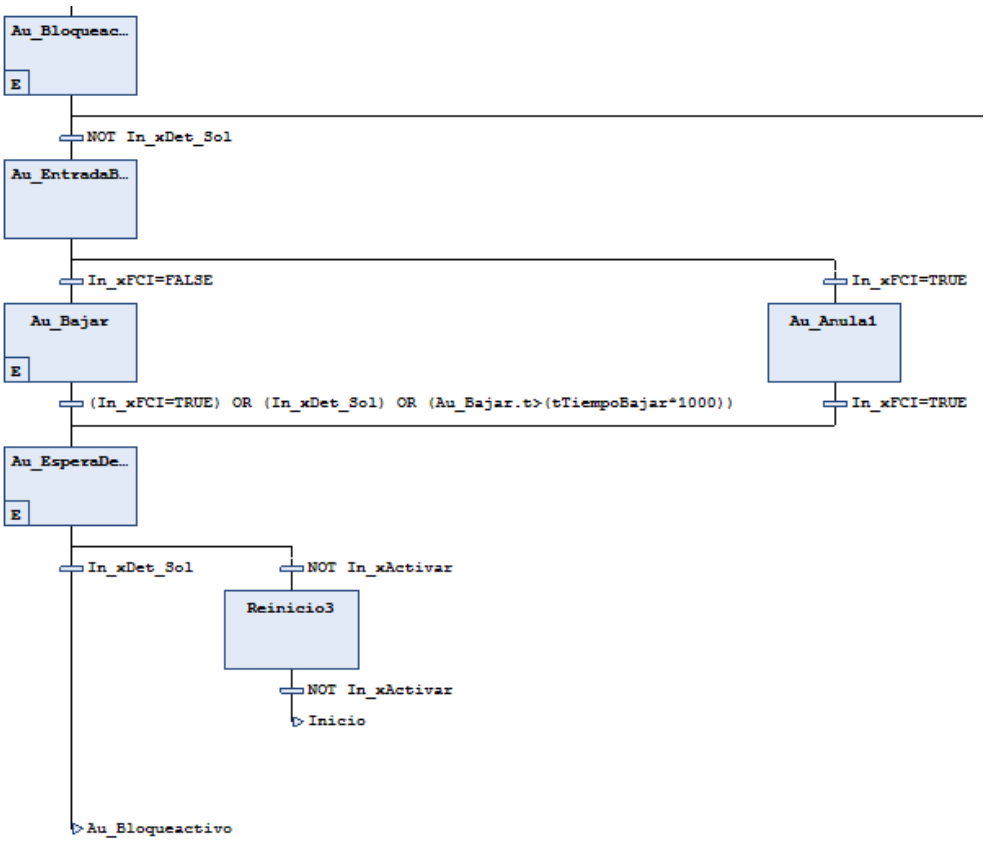

Figura 3-43 - Código activación automática

Como en este modo no hay más que una entrada con solo 2 estados, el modo de funcionamiento es similar al manual, solo que después de ejecutar una acción de subida o bajada se queda retenido en la fase *Au_EsperaDesact*. De esta forma cuando el detector cambie de estado pasa directamente a la otra vertiente, se ejecuta al momento y queda retenido en otra fase de desactivación.

3.7.- Bloque funcional de reloj

Este bloque adicional surge por la necesidad de proporcionar la hora actual a los demás bloques de función que tengan algún tipo de control horario, luces, riego y climatización en este caso. De todos modos, también se puede utilizar para mostrar la fecha y hora en algún panel de la instalación si fuera necesario.

Figura 3-44 - Representación del bloque reloj

3.7.1.- Entradas/Salidas

Este bloque no cuenta con entradas de ningún tipo, únicamente cuenta con 2 salidas para mostrar fecha y hora para utilizarlos en algún reloj o como entrada para otros bloques.

Variable	Descripción
Out_dtaFecha_Actual	Salida con formato fecha
Out_todHora_Actual	Salida con formato hora

Tabla 3-8 - E/S bloque reloj

3.7.2.- Código

Se aprovechan una serie de funciones procedentes de la biblioteca *SysTimeRtc*, que permiten obtener la hora y fecha en varios formatos. Las funciones utilizadas son:

- **SysTimeRtcGet:** permite obtener los segundos pasados desde 1970.
- **SysTimeRtcConvertUtcToLocal:** esta función modifica los segundos obtenidos en la función anterior, para que se adapten a la zona horaria local.
- **UDINT_TO_DT:** esta función transforma los segundos obtenidos en la función anterior al formato fecha y hora.

- **DT_TO_DATE:** esta función transforma el formato obtenido en la función anterior al formato de únicamente fecha.

- **DT_TO_TDO:** esta última función transforma el formato obtenido de la función *UDINT_TO_DT* a solo hora para utilizarlo en los demás bloques.

```
udiSegundos_1970 := SysTimeRtcGet(udiResult);
IF udiResult <> 0 THEN
    RETURN;
END_IF

udiResult := SysTimeRtcConvertUtcToLocal(udiSegundos_1970,udiSegundos_1970_Local);
IF udiResult <> 0 THEN
    RETURN;
END_IF

dtFechaYHora_Actual := UDINT_TO_DT(udiSegundos_1970_Local);

datFecha_Actual := DT_TO_DATE(dtFechayHora_Actual);

todHora_Actual := DT_TO_TOD(dtFechayHora_Actual);

Out_datFecha_Actual:=datFecha_Actual;

Out_todHora_Actual:=todHora_Actual;
```

Figura 3-45 - Código reloj

4. Visualizaciones

Junto con los bloques funcionales, en la biblioteca se han incluido además unas sencillas visualizaciones para acompañarlos. La versión de Codesys 3.5 en la que se ha desarrollado el proyecto permite tratar a las visualizaciones como bloques funcionales, y de este modo poder añadirlos a la biblioteca.

Estos bloques además pueden configurarse para que al añadirse a la visualización de algún proyecto presenten un formulario en el que se pueden asociar variables cuando se coloquen por primera vez, de esta forma se pueden adaptar a las variables que utilice un nuevo usuario.

Se procede a describir cómo se crean las visualizaciones y más adelante explicar cada una de las que incluye la biblioteca, indicando cada uno de sus elementos, y el aspecto que presentan cuando se encuentran en funcionamiento.

4.1.- Creación de una visualización

El proceso para crear una visualización es relativamente simple, y al igual que los bloques funcionales, se puede desarrollar en un proyecto y probarla, para más tarde copiarla directamente en la biblioteca y configurarla.

Se comienza por añadir un objeto nuevo, sobre *Application* de la siguiente forma:

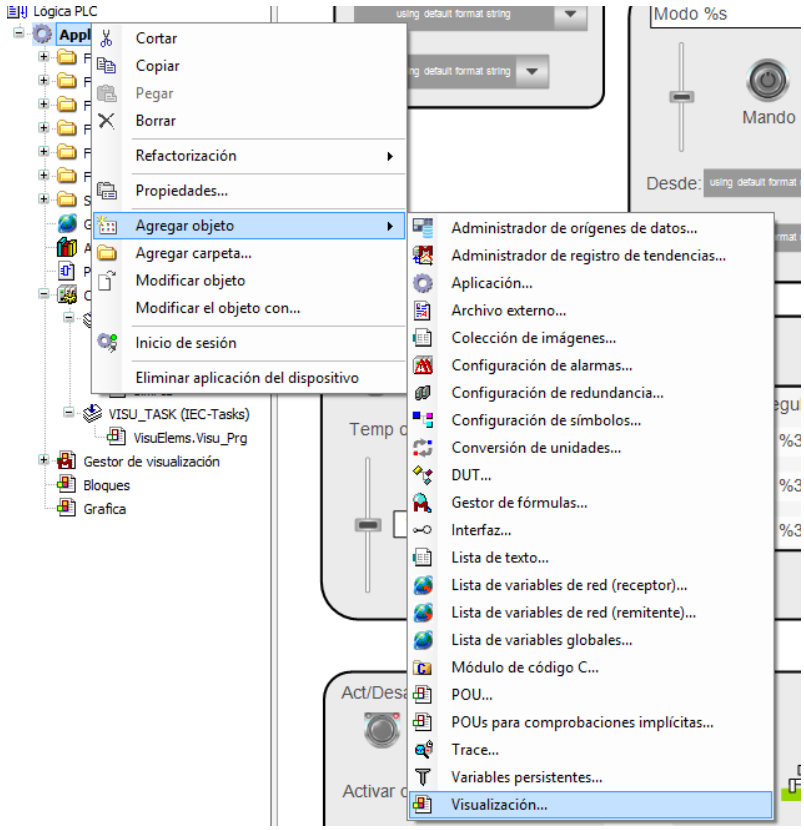

Figura 4-1 - Añadir objeto visualización

En este nuevo objeto, la pestaña de la derecha cambia a una forma nueva, donde se pueden ver los diferentes elementos visuales que se pueden colocar ordenados por bibliotecas, desde las que vienen por defecto en todos los proyectos, a las bibliotecas de interfaz que se añadan.

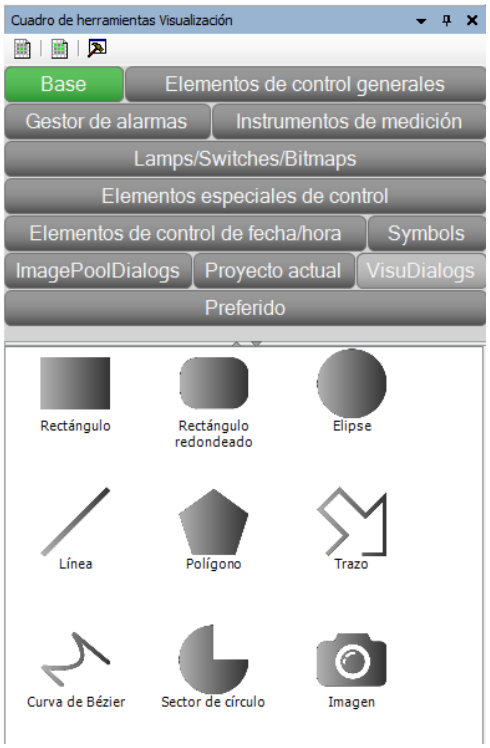

Figura 4-2 - Ventana de elementos

Al añadirse un objeto, en esa misma pestaña se pueden observar y alterar sus propiedades siempre que ese objeto lo permita, desde forma, color, textos, visibilidad etc. Además, es posible asociar variables a estas características para que alteren estas propiedades en función de lo que esté pasando en el programa.

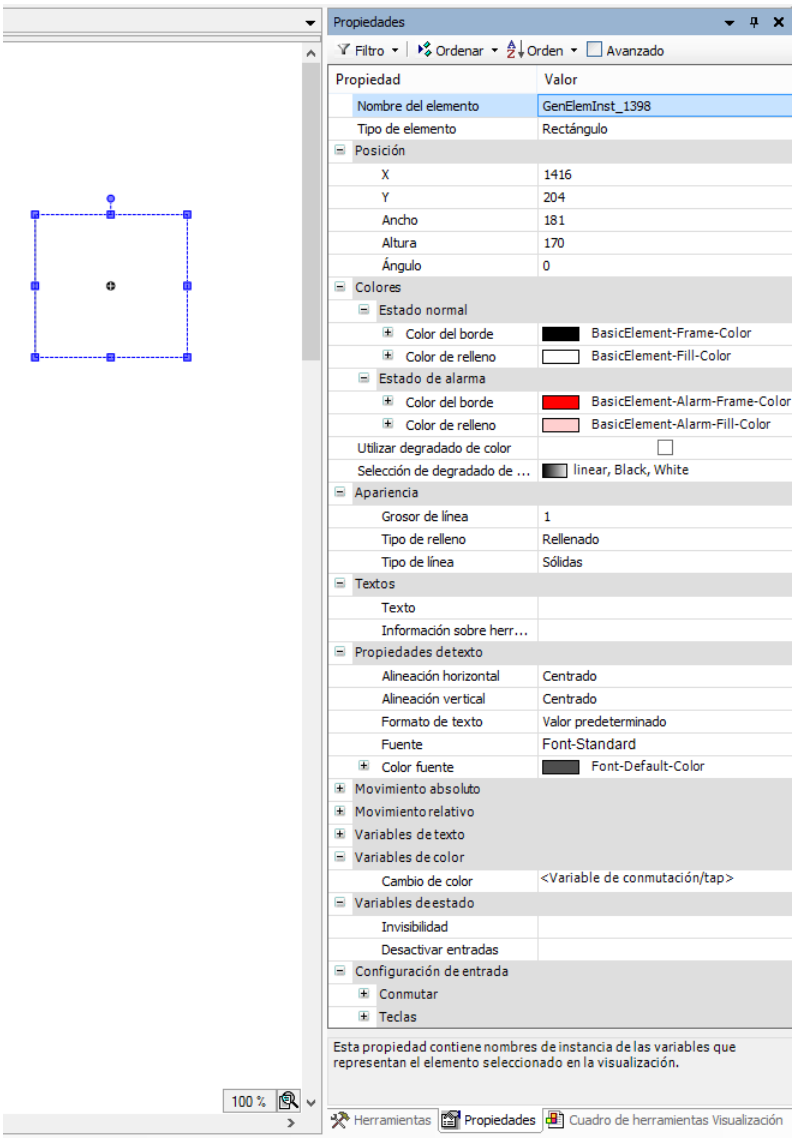

Figura 4-3 - Opciones de edición de elemento

Añadiendo elementos de esta forma se pueden acabar haciendo simulaciones bastante complejas y funcionales, pero si se quieren incluir en una biblioteca, para poder usarse de forma correcta se debe dar un paso más.

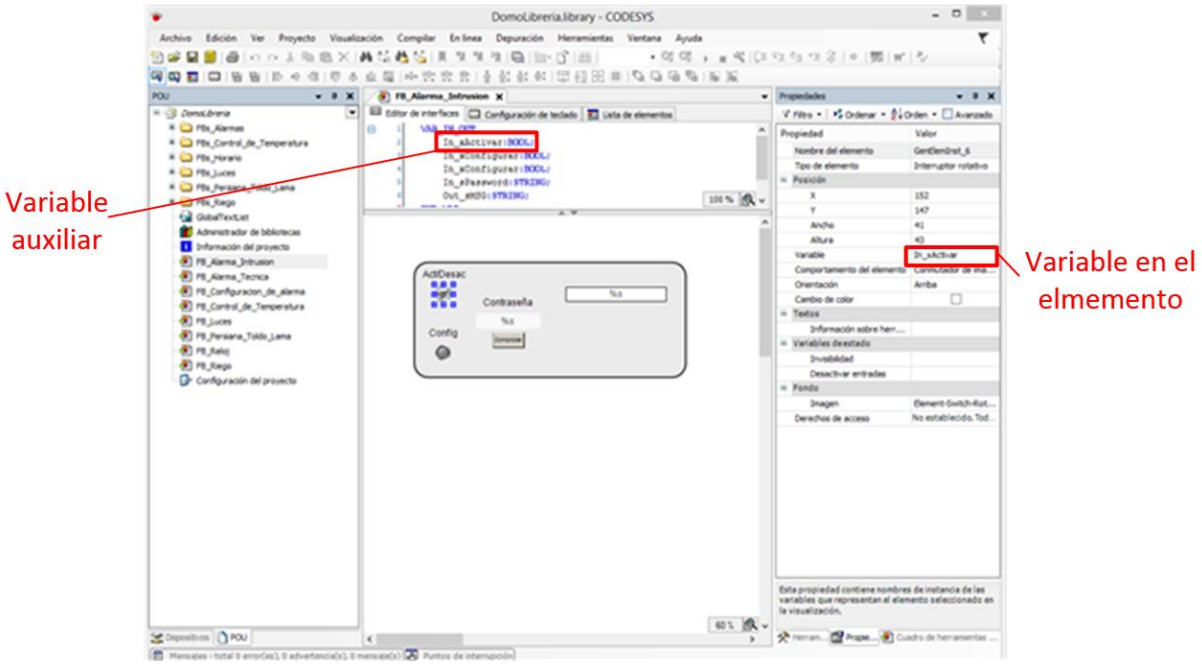

Figura 4-4 - Variables auxiliares

Como se ve en la figura anterior, el botón de activación lleva asociado una variable auxiliar creada dentro de la visualización. Este paso es el que hace que, al utilizar esta visualización en otro proyecto, se llame a un formulario de configuración donde el nuevo usuario puede asignar su variable en ese lugar concreto de la visualización. De esta forma es posible crear visualizaciones y dejarlas preparadas para usos de otros usuarios futuros.

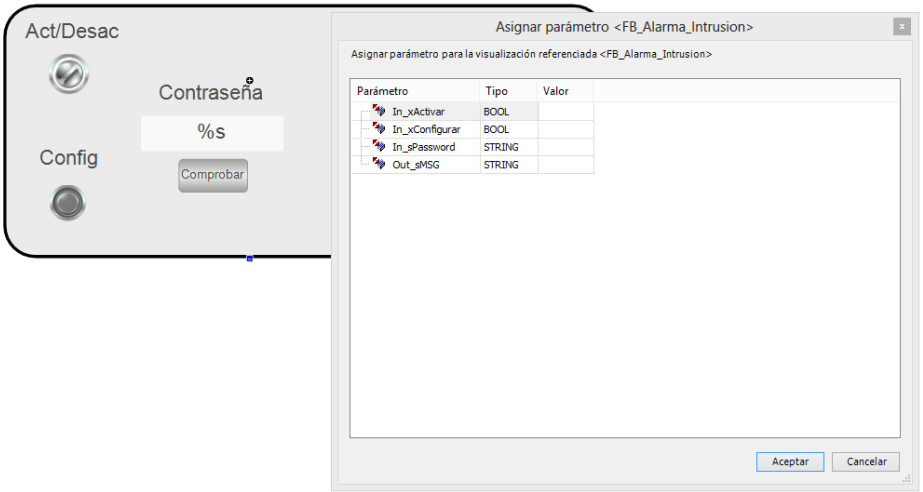

Figura 4-5 - Añadir visualización

Por conveniencia y para hacer más sencillo identificar y colocar las variables a posteriori, las visualizaciones y los bloques funcionales creados comparten los mismos nombres en todas sus variables. Por ejemplo, en el bloque de luces *In_iSel_Modo* es utilizado para seleccionar el modo de funcionamiento, en el bloque de visualizaciones también existe *In_iSel_Modo* asociado a un selector, de este modo si se asocia la misma variable global en estos dos lugares, el programa y la visualización funcionarán correctamente.

4.2.- Luces

Para la visualización de luces se han añadido todos los elementos de control posibles, cambiando de un modo a otro con un selector para poder utilizarlos todos con un solo panel si es necesario.

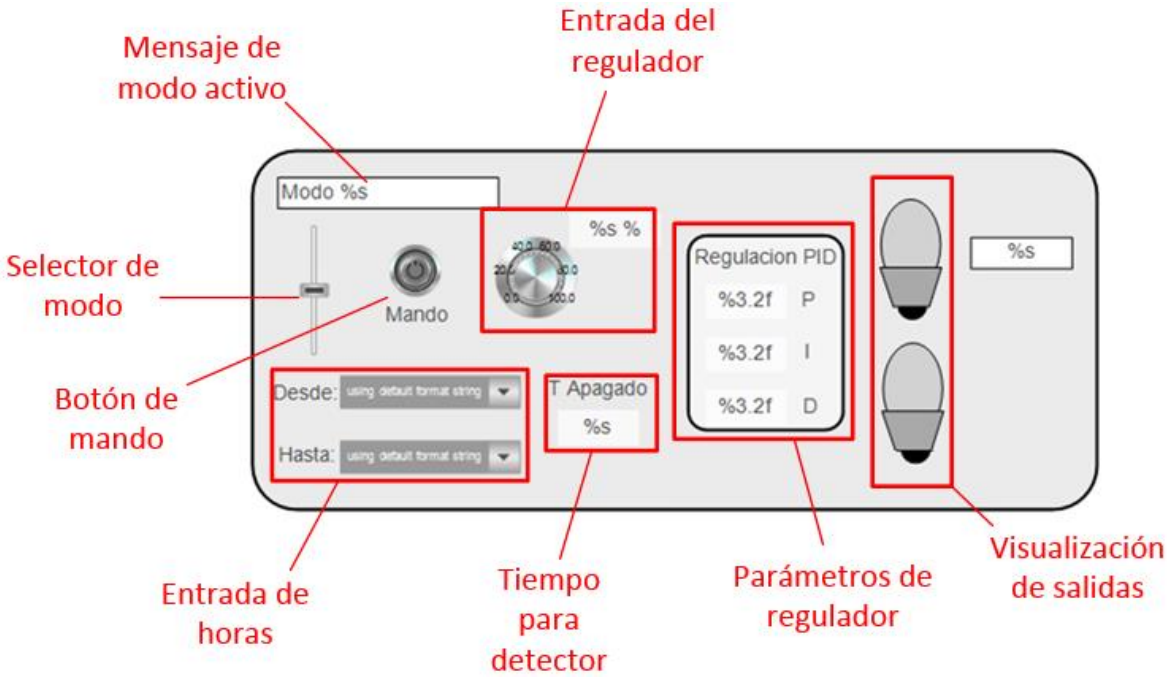

Figura 4-6 - Visualización de luces

A continuación, una vez colocada en un proyecto, utilizándose el modo 6:

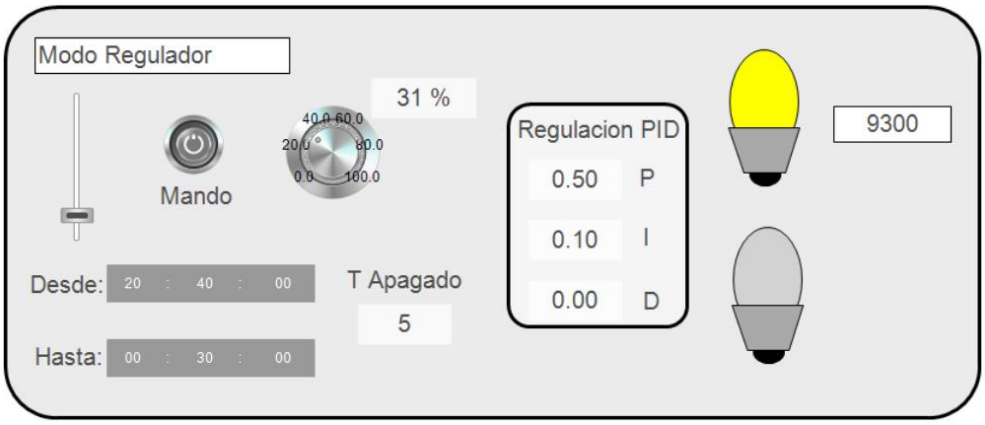

Figura 4-7 - Visualización de luces en funcionamiento

4.3.- Climatización

Por último, la visualización para el control de climatización es bastante grande para alojar todos los elementos necesarios como: controles de activación, selector de temperaturas, configuración del PID, mensajes para informar al usuario…etc. Al igual que el riego, las entradas para las horas se ocultan si no se está en modo horario.

La salida cuenta con cambios de color para representar si está calentando, enfriando o detenido, además de un campo para mostrar el porcentaje de potencia que está siendo utilizada. Además, también cuenta con una escala para visualizar la temperatura en ese momento, o leerse directamente de un campo donde se muestra.

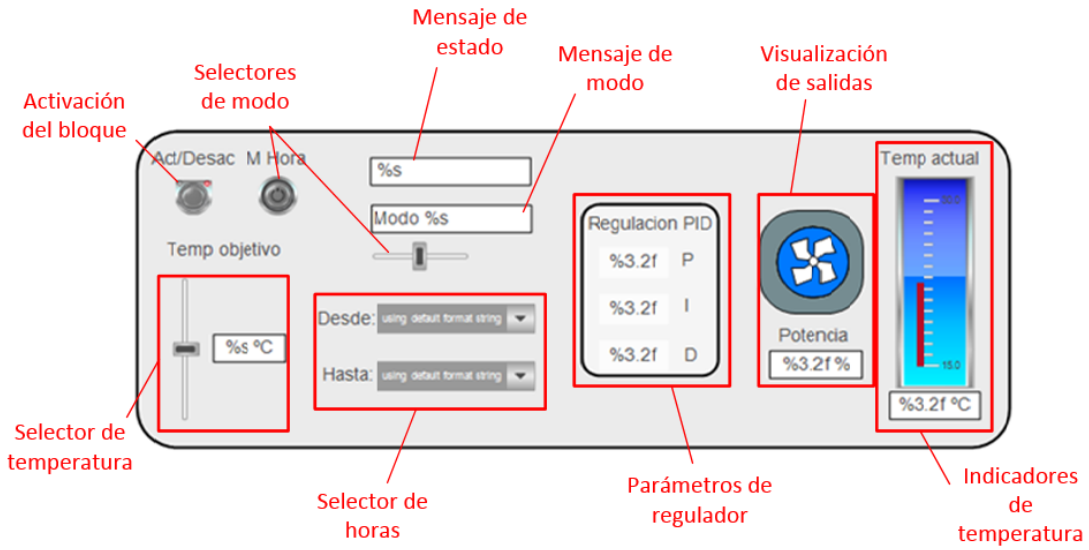

Figura 4-8 - Visualización de climatización

A continuación, una vez colocada en un proyecto, mientras está calentando:

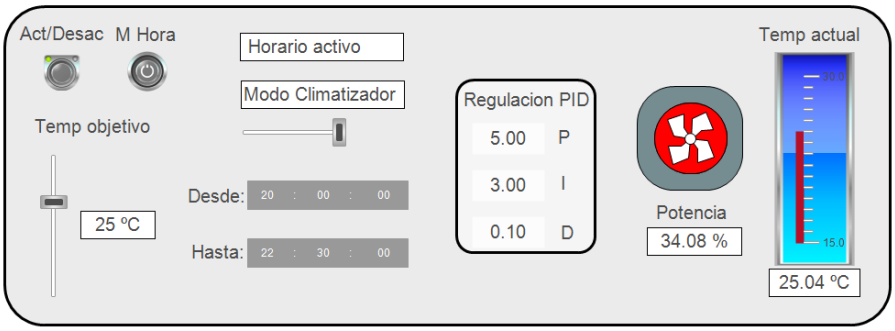

Figura 4-9 - Visualización de climatización en funcionamiento

4.4.- Alarmas

Al igual que los bloques funcionales de alarmas están separados en según su finalidad, las visualizaciones siguen un diseño similar y existe una correspondiente a cada uno de esos bloques.

4.4.1.- Técnicas

Para el bloque de alarmas técnicas se ha separado el control de válvulas que siempre está "activo" mientras no haya una alarma, del panel de control de alarmas.

La representación de las válvulas cambia de color según el estado en que se encuentren, con un visible tono rojo indicando que se encuentran cerradas.

En la parte de control de alarmas, se dispone de un campo para mostrar mensajes donde se ve el estado del bloque, y pilotos luminosos para conocer el estado de los sensores.

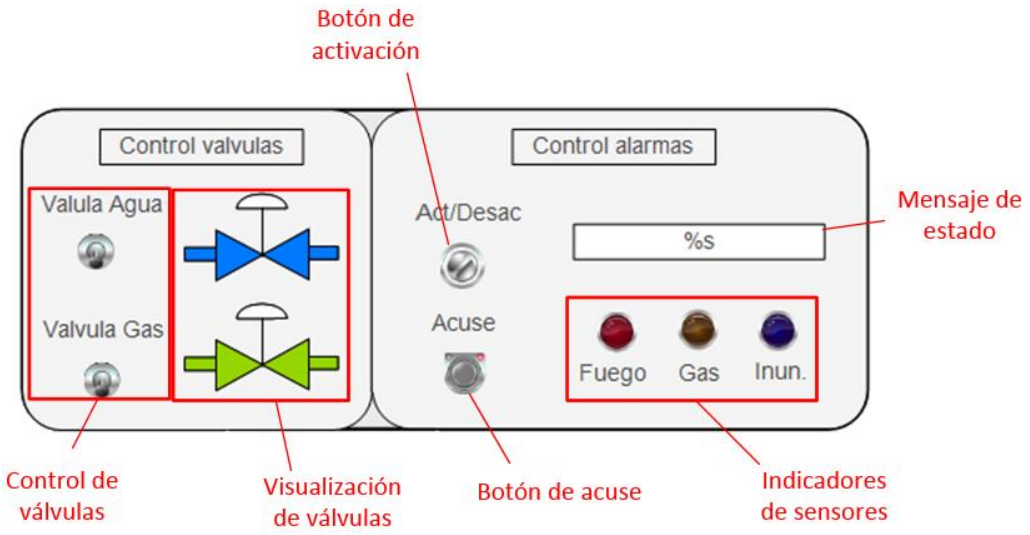

Figura 4-10 - Visualización de técnicas

A continuación, una vez colocada en un proyecto, con una alarma de inundación activa:

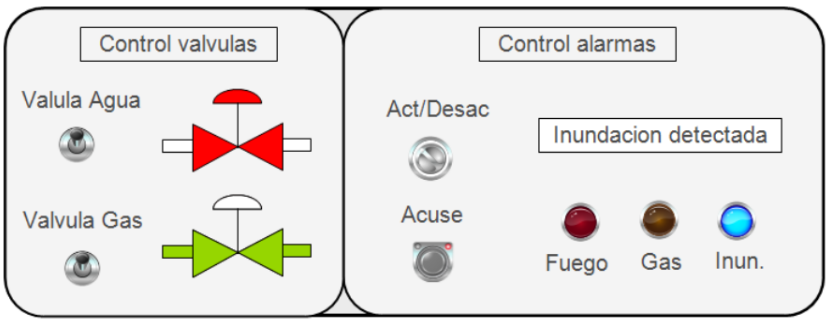

Figura 4-11 - Visualización de técnicas en funcionamiento

4.4.2.- Intrusión

El control de intrusión es más sencillo, contando con los diferentes botones de control, el campo donde se introduce la contraseña y el campo de mensaje para indicar al usuario que está pasando o que debe hacer.

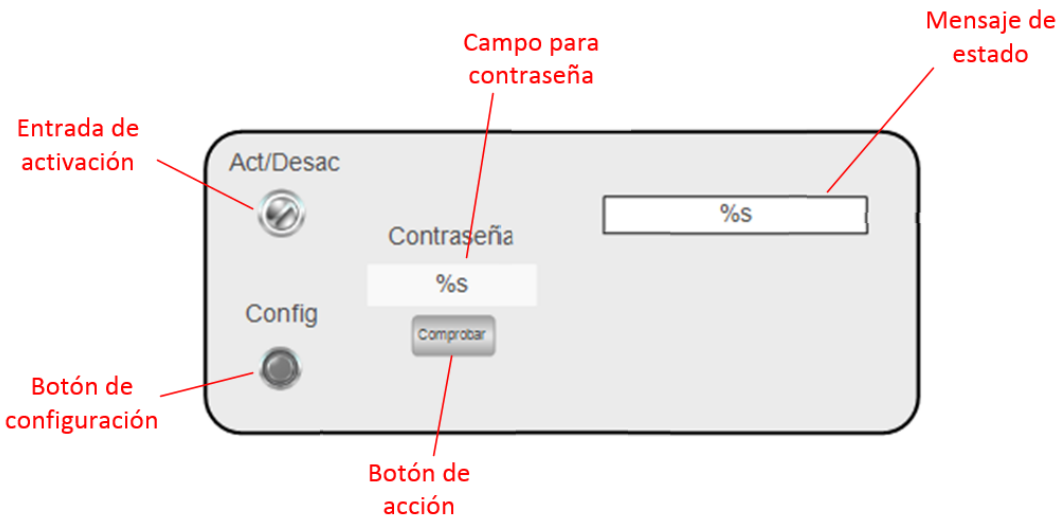

Figura 4-12 - Visualización de intrusión

A continuación, una vez colocada en un proyecto, con la alarma activada:

Figura 4-13 - Visualización de intrusión en funcionamiento

4.4.3.- Configuración

Para el bloque de configuración de secuencia, se tienen selectores para las opciones de activación/desactivación y el modo de funcionamiento.

También cuenta con representaciones del piloto y la sirena, que cambian de color según el estado en el que se encuentren.

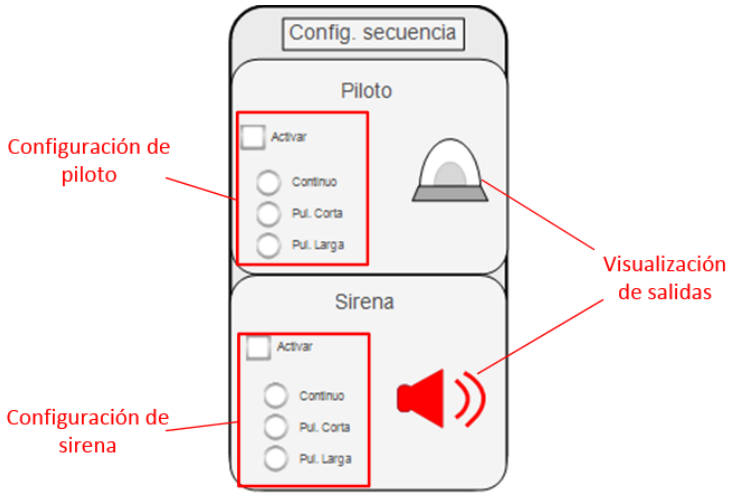

Figura 4-14 Visualización de configuración

A continuación, una vez colocada en un proyecto, mientras se produce una alarma:

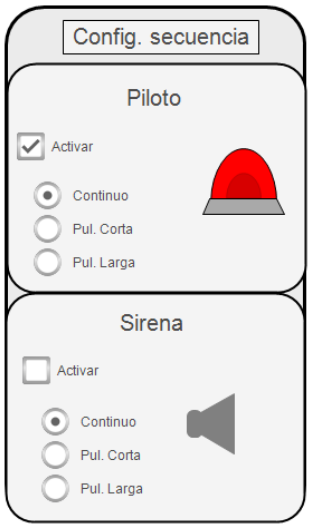

Figura 4-15 - Visualización de configuración en funcionamiento

4.5.- Riego

Para el bloque de riego, además de todos los elementos de control esperables, también se ha añadido otra funcionalidad más. Si no se tiene activado el modo horario, las opciones de introducir hora y humedad quedan ocultas para una interfaz más limpia.

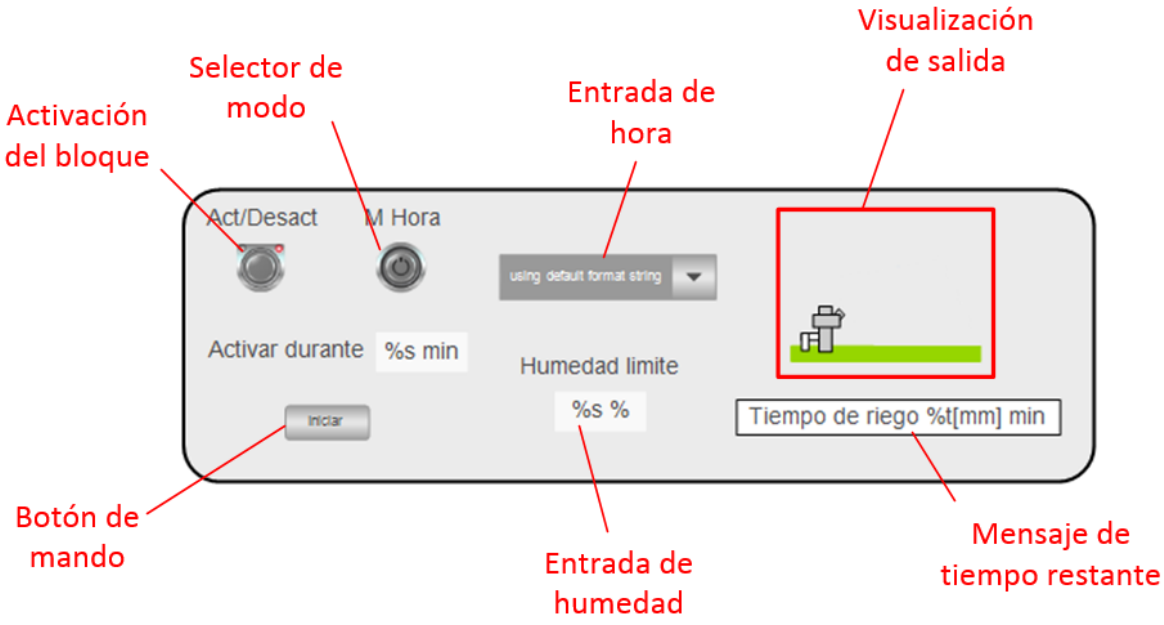

Figura 4-16 - Visualización de riego

A continuación, una vez colocada en un proyecto, una vez se ha activado el riego en modo automático:

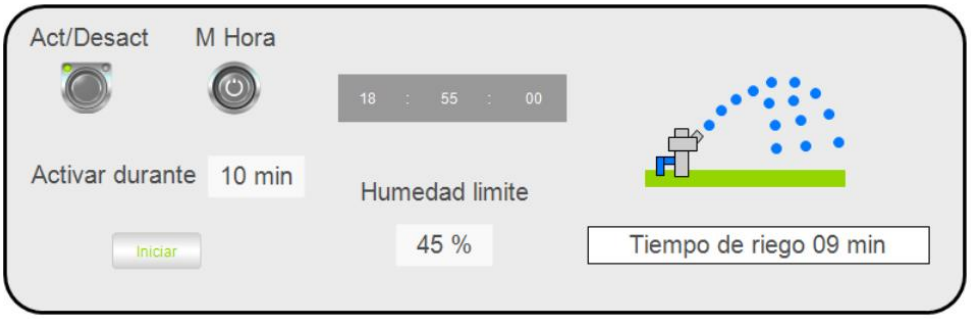

Figura 4-17 - Visualización de riego en funcionamiento

4.6.- Motores

La visualización del bloque para motores cuenta con flechas direccionales que indican la dirección de movimiento, estas cambian de intensidad en función de si están activas o no en ese momento.

También cuenta con un campo con el que puede verse el modo de funcionamiento activo, e indicaciones que aparecerán en caso de que los finales de carrera se activen.

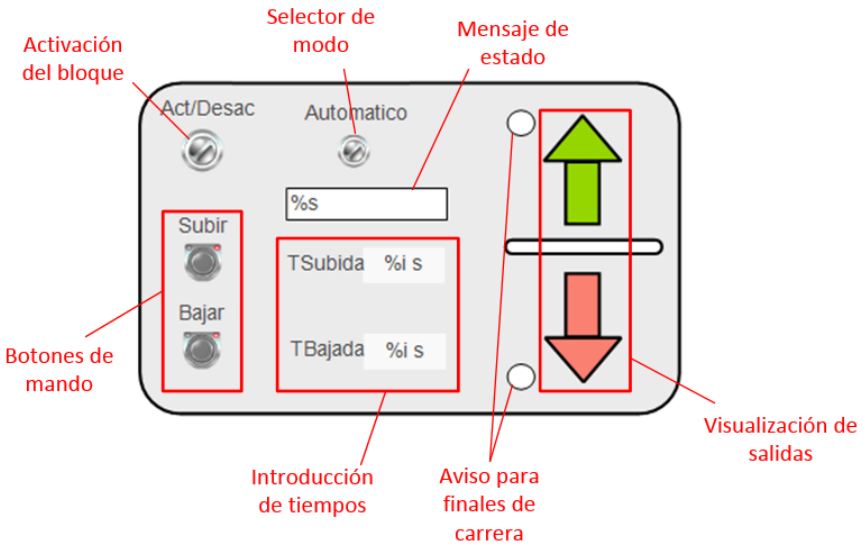

Figura 4-18 - Visualización de motores

A continuación, una vez colocada en un proyecto, mientras se da la acción de subir en modo manual:

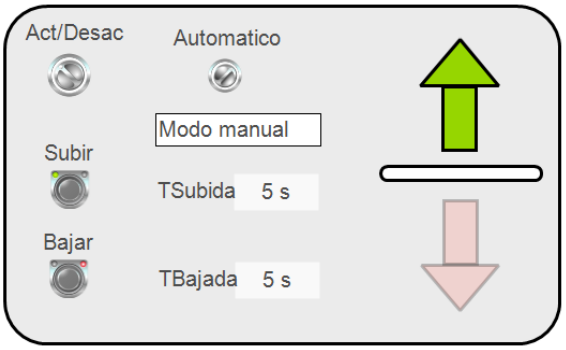

Figura 4-19 - Visualización de motores en funcionamiento

4.7.- Reloj

Se trata de una pequeña visualización para el bloque con el mismo nombre. Consta únicamente de 2 variables, para fecha en la parte superior y para hora en la inferior.

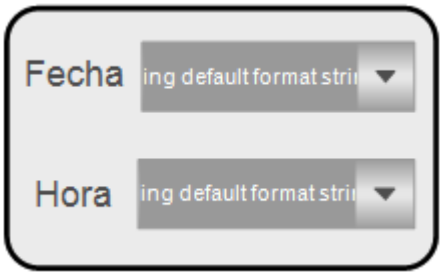

Figura 4-20 - Visualización de reloj

A continuación, una vez colocada en un proyecto:

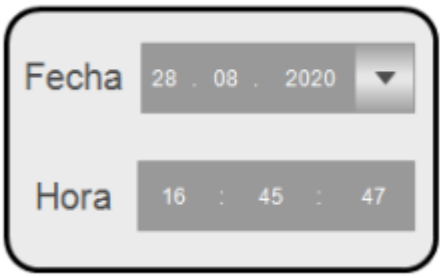

Figura 4-21 - Visualización de reloj en funcionamiento

4.8.- Versiones reducidas

Debido al funcionamiento de Codesys, si una visualización de una biblioteca ha sido preparada con variables auxiliares para asignar en futuros usos, es imperativo que todas esas variables queden cubiertas al utilizarse en un proyecto o presentará errores de compilación.

Esto unido a que en los apartados siguientes se pretende realizar una plantilla y una simulación utilizando la biblioteca, implica que, si se hace una programación parcial de alguno de sus bloques, no se incluyen todas las variables necesarias para que las visualizaciones funcionen en su estado actual. Por ello se incluyen visualizaciones con

menos elementos, pensadas para no utilizar ciertas funciones o solo realizar una de ellas. Estas incluyen cambios en:

- **Luces:** se incluye una visualización para el modo detector, interruptor, horario y regulador.
- **Climatización:** se incluye una visualización con solo modo manual.
- **Configuración de alarmas:** se incluye una visualización para tener solo sirena y otra para solo piloto.
- **Motores:** se incluye una visualización que solo tiene modo automático, otra con solo finales de carrera y otra con solo modo manual.

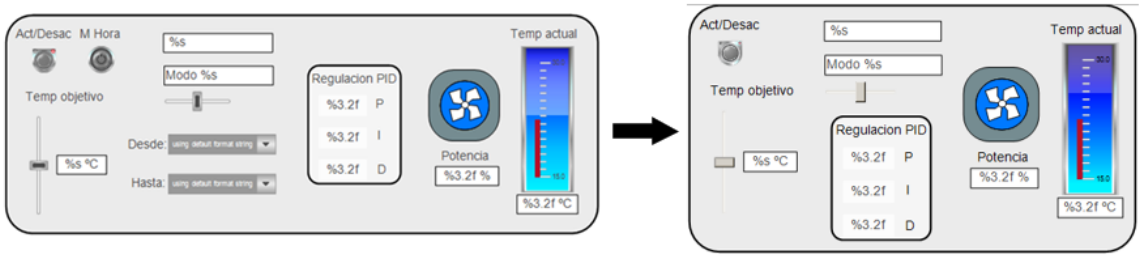

Figura 4-22 – Ejemplo de panel reducido

De forma adicional, para poder visualizar el estado de los diferentes sistemas sin saturar la pantalla con todos los bloques y su configuración, también se incluyen visualizaciones únicamente de sus elementos de salida, siendo los mismos que los representan en un bloque completo. Aquí se incluyen:

- Bombillas
- Representación del climatizador con la temperatura
- Válvulas de gas y agua
- Sirena y piloto
- Aspersor

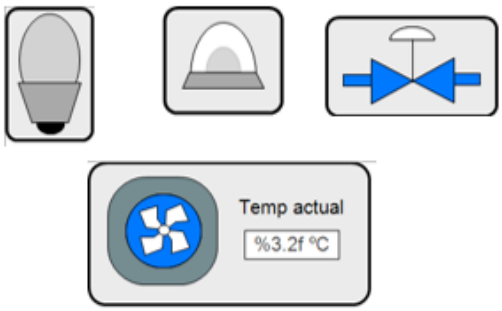

Figura 4-23 – Ejemplo de algunos elementos

5. Plantilla Excel

Para facilitar las cosas a alguien que no tenga mucha experiencia programando con Codesys, se ha desarrollado e incluido también una plantilla hecha en Excel. Se ha realizado sobre la versión 2004 de Excel con el nombre **PlantillaDomotica.xlms**.

Figura 5-1 - Acerca de Excel

La plantilla permite elegir cierto número de bloques y elementos de los mismos a incluir en el programa, con algunas ayudas para su configuración y el nombre de las variables que el usuario quiera utilizar.

Al final de esta plantilla se puede obtener el texto necesario a incluir en el archivo de variables globales, las llamadas a los bloques funcionales en el programa principal y la programación necesaria en lenguaje ST (*Structured Text*).

La plantilla tiene capacidad máxima para incluir variables y configuraciones para:

- 1 bloque reloj
- 4 bloques de luces
- 1 set de bloques de alarma
- 1 bloque de riego
- 2 bloques de motores
- 2 bloques de climatización

A través del uso de macros se puede alterar que bloques incluir en la configuración eliminando el código innecesario. El archivo consta de 5 pestañas que se describen en los apartados siguientes.

5.1.- Introducción

Esta primera pestaña sirve únicamente como guía para utilizar la plantilla, contiene información resumida sobre las demás pestañas y el orden en el que deberían visitarse o modificar su contenido.

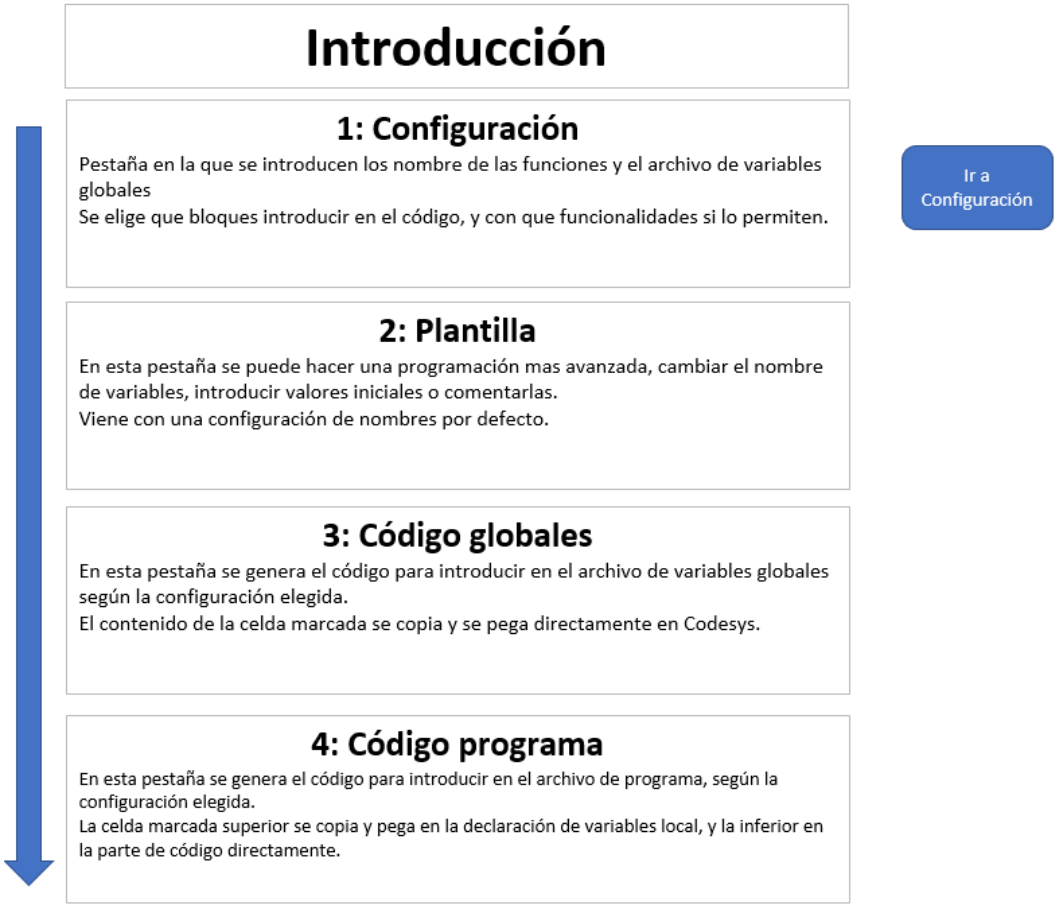

Figura 5-2 - Pestaña introducción

5.2.- Configuración

En esta pestaña se pueden elegir los bloques funcionales que se quieren incluir en el proyecto a desarrollar, además de elegir si utilizar o no ciertas funciones de los mismos.

También en esta pestaña, se les da nombre a esos bloques y al archivo de variables globales, que se incluirán de forma automática en el código que se genere en las próximas pestañas.

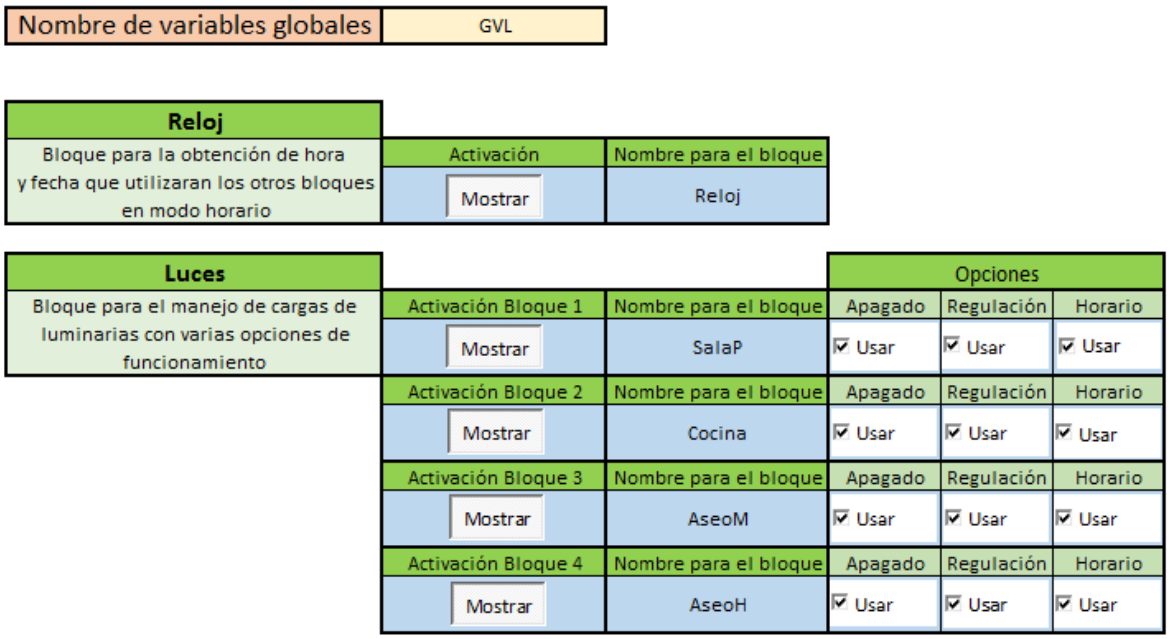

Figura 5-3 - Extracto de pestaña Configuración

Al utilizar el botón Generar, se actualizarán las pestañas siguientes con la configuración elegida, ocultando por completo las líneas de código asociadas a las funciones o bloques que se haya elegido no utilizar.

5.3.- Plantilla

Si se desea hacer algún tipo de configuración más avanzada, es posible entrar a esta pestaña para cambiar nombres de variables, asignar valores iniciales o añadir comentarios.

En primera instancia, las variables ya cuentan con una serie de nombres predefinidos para identificarlas de una forma más inmediata si fuera necesario utilizar las visualizaciones en Codesys.

Reloj	E/S Bloque	Nombre de variable	Valor Inicial	Comentarios
	Out_datFecha_Actual	O_datFecha_Actual		
	Out_todHora_Actual	O_todHora_Actual		

Luces	E/S Bloque	Nombre de variable	Valor Inicial	Comentarios
	In_iSel_Modo	I_iModoLuz	1	
	In_xMando	I_xMandoLuz		
	In_Tapagado	I_iTApagado	5	
	In_iIntensidadRegulador	I_iIntensidad		
	In_iSensorLuminosidad	I_iSensorLuminosidad		
	In_todReloj	O_todHora_Actual		
	In_todHoraI	I_todHoraL1		
	In_todHoraF	I_todHoraL2		
	In_rPID_KP	I_rPropL	0.5	
	In_rPID_TN	I_rIntL	0.1	
	In_rPID_TV	I_rDerL	0	
	Out_xConmutar1	O_xConmutar1		
	Out_xConmutar2	O_xConmutar2		
	Out_iLuminosidad	O_iLuminosidad		
	Out_sMSGModo	O_sModoLuz		

Figura 5-4 - Extracto de pestaña Plantilla

Cada una de las casillas, E/S Bloque, coincide con el nombre de la entrada o salida del bloque funcional de la biblioteca que representa.

Los campos azules, hacen referencia a elementos que aparecerán más tarde en el código generado, mientras que los grises hacen referencia a variables que dependen de los valores que se les dé en otro bloque, por lo tanto, estas no deberían modificarse si no se sabe lo que se está haciendo.

En caso de querer modificar algún valor es importante saber que Codesys no admitirá 2 nombres iguales, espacios, o caracteres especiales como la “ñ”.

5.4.- Variables globales

Esta pestaña contendrá el texto generado que se ha de incluir en archivo de programa en la parte de declaración de variables globales, el que Codesys nombra por defecto GVL.

Una vez generado se encontrarán las variables separadas por bloques, con un comentario que incluye el nombre elegido para su bloque. Además, el nombre del bloque también estará incluido automáticamente al final de cada una de las variables de la forma, "Variable_Nombre de bloque" para identificarlas fácilmente según al bloque que pertenecen.

Lo que se debe copiar y pegar es lo que este dentro de las líneas delimitantes como se ve en la siguiente figura.

```
VAR_GLOBAL

//BloqueReloj

O_datFecha_Actual_Reloj:DATE;
O_todHora_Actual_Reloj:TOD;

//BloqueSalaP

I_iModoLuz_SalaP:INT:=1;
I_xMandoLuz_SalaP:BOOL;
I_iTApagado_SalaP:INT:=5;
I_iIntensidad_SalaP:INT;
I_iSensorLuminosidad_SalaP:INT;
I_todHoraL1_SalaP:tod;
I_todHoraL2_SalaP:tod;
I_rPropL_SalaP:REAL:=0.5;
I_rIntL_SalaP:REAL:=0.1;
I_rDerL_SalaP:REAL:=0;
O_xConmutar1_SalaP:BOOL;
O_xConmutar2_SalaP:BOOL;
O_iLuminosidad_SalaP:INT;
O_sModoLuz_SalaP:STRING;
```

Figura 5-5 - Extracto de la pestaña Código globales

5.5.- Código programa

Esta pestaña compone el código para el archivo de programa, el que al seleccionar un proyecto estándar en Codesys se nombra PLC_PRG por defecto.

Se divide en 2 partes, la primera es la configuración de variables a incluir en la parte superior del programa para llamar a las instancias de los bloques para su uso, y la segunda el propio programa ya generado.

De una forma similar a la pestaña Variables globales, incluye las variables con el nombre del bloque asociado al final.

Lo que se debe copiar y pegar es lo que se encuentra dentro de las líneas delimitantes al igual que el código de variables globales como se ve en las siguientes figuras.

```
Var

Reloj:Domotica.FB_Reloj;
SalaP:Domotica.FB_Control_de_Luces;
Cocina:Domotica.FB_Control_de_Luces;
AseoM:Domotica.FB_Control_de_Luces;
AseoH:Domotica.FB_Control_de_Luces;
Intrusion:Domotica.FB_Alarma_Intrusion;
Tecnicas:Domotica.FB_Alarma_Tecnica;
Config:Domotica.FB_Configuracion_de_Alarma;
Riego:Domotica.FB_CRiego;
Toldo:Domotica.FB_Cpersiana_Toldo_Lama;
Cierre:Domotica.FB_Cpersiana_Toldo_Lama;
ClimaSala:Domotica.FB_Control_de_Temperatura;
ClimaEmple:Domotica.FB_Control_de_Temperatura;

End_Var
```

Figura 5-6 - Extracto de Código programa 1

```
Reloj           (
                Out_datFecha_Actual=>GVL.O_datFecha_Actual_Reloj,
                Out_todHora_Actual=>GVL.O_todHora_Actual_Reloj,
                );

SalaP           (
                In_iSel_Modo:=GVL.I_iModoLuz_SalaP,
                In_xMando:=GVL.I_xMandoLuz_SalaP,
                In_Tapagado:=GVL.I_iTApagado_SalaP,
                In_iIntensidadRegulador:=GVL.I_iIntensidad_SalaP,
                In_iSensorLuminosidad:=GVL.I_iSensorLuminosidad_SalaP,
                In_todReloj:=GVL.O_todHora_Actual_Reloj,
                In_todHoraI:=GVL.I_todHoraL1_SalaP,
                In_todHoraF:=GVL.I_todHoraL2_SalaP,
                In_rPID_KP:=GVL.I_rPropL_SalaP,
                In_rPID_TN:=GVL.I_rIntL_SalaP,
                In_rPID_TV:=GVL.I_rDerL_SalaP,
                Out_xConmutar1=>GVL.O_xConmutar1_SalaP,
                Out_xConmutar2=>GVL.O_xConmutar2_SalaP,
                Out_iLuminosidad=>GVL.O_iLuminosidad_SalaP,
                Out_sMSGModo=>GVL.O_sModoLuz_SalaP,
                );
```

Figura 5-7 - Extracto de Código programa 2

5.6.- Macros

Por último, se explica de una forma sencilla las macros utilizadas para hacer funcionar la plantilla en Excel, asociadas todas ellas a la pestaña de configuración.

En esencia existen 2 tipos, una para cambiar el texto al hacer click que acompaña los diferentes botones y opciones, y otra que oculta o muestra líneas enteras de celdas en función de los valores de esos botones y opciones seleccionadas.

Lo primero, es añadir las herramientas de programador en Excel que permiten la colocación de elementos utilizables para macros, ya que por defecto no suelen estar activas. Esto se consigue pulsando el botón derecho del ratón sobre la barra de herramientas, y en el menú que se abre seleccionar *Personalizar la cinta de opciones.* Al hacer esto se abre la ventana que se muestra en la figura siguiente en la que ya se puede seleccionar mostrar las herramientas de programador que se añadirán a la barra de herramientas bajo el nombre *Programador*.

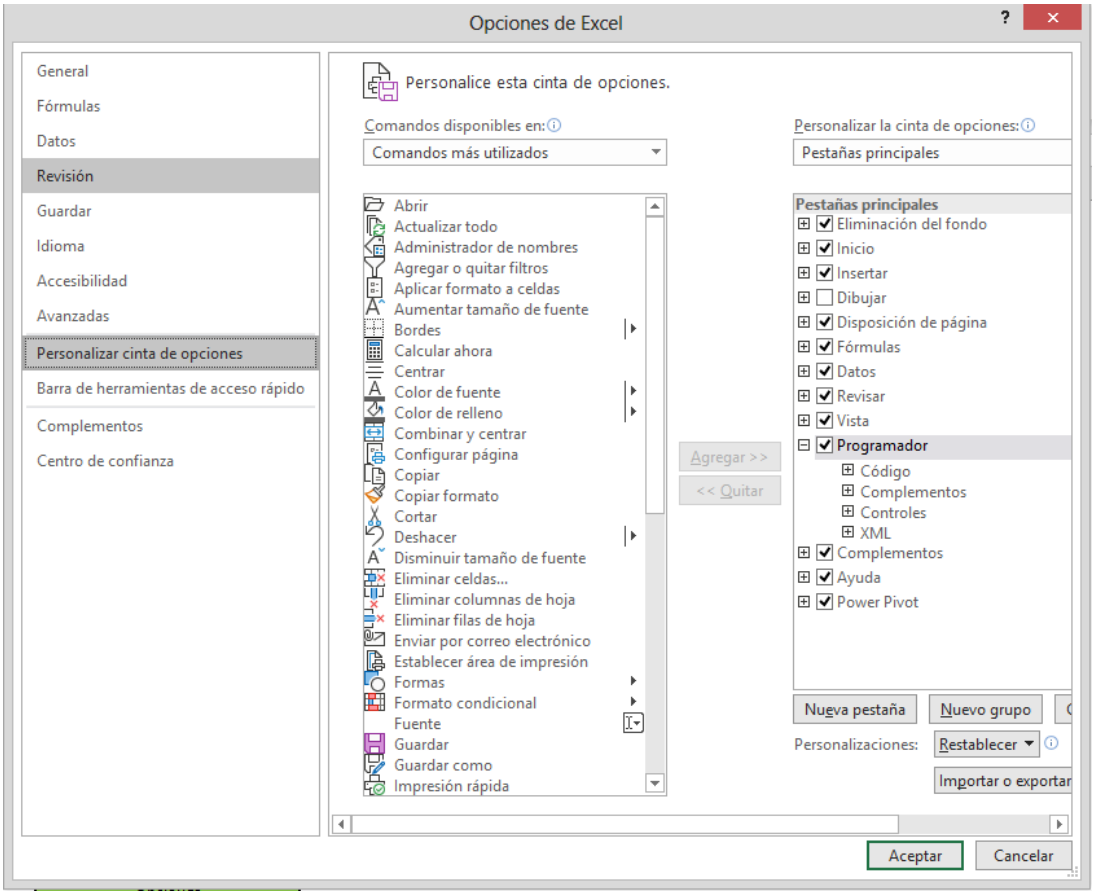

Figura 5-8 - Añadir herramientas en Excel

Con estas nuevas herramientas se pueden utilizar los botones para utilizar macros.

- **Nombres:** al haber colocado un botón desde las herramientas de programador, con click derecho se pueden alterar sus propiedades y cambiar su nombre. Este nombre será el mismo que el de la función que se ejecutara al apretarlo, de este modo como se ve en la captura siguiente, el "caption" que es el texto que acompaña al botón, se altera cada vez que cambia el valor de ese botón.

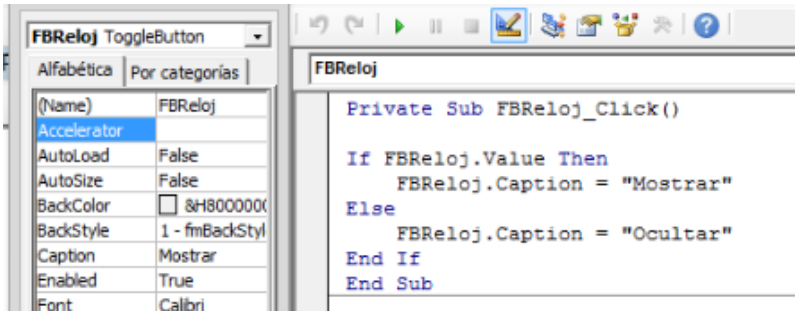

Figura 5-9 - Ejemplo 1 de macro

- **Mostrar/Ocultar filas:** con una filosofía similar, a ese mismo botón se le asocia otra función diferente, esta vez para mostrar u ocultar filas en otras pestañas.

  Esto se consigue con las líneas de código de la siguiente figura, que cambian el valor "Hidden" de las filas o conjunto de filas deseadas según el estado del botón o la opción seleccionada correspondiente.

```
If FBReloj.Value Then
    Application.Worksheets("Plantilla").Rows("8:10").Hidden = False
    Application.Worksheets("Código globales").Rows("4:7").Hidden = False
    Application.Worksheets("Código programa").Rows("6:6").Hidden = False
    Application.Worksheets("Código programa").Rows("28:31").Hidden = False
Else
    Application.Worksheets("Plantilla").Rows("8:10").Hidden = True
    Application.Worksheets("Código globales").Rows("4:7").Hidden = True
    Application.Worksheets("Código programa").Rows("6:6").Hidden = True
    Application.Worksheets("Código programa").Rows("28:31").Hidden = True
End If
```

Figura 5-10 - Ejemplo 2 de macro

A fin de evitar que esto se ejecute continuamente, todas las macros que tienen esta finalidad se encuentran dentro de un bucle sujetas al valor del botón "Generar" en la parte superior de la pestaña Configuración, de este modo únicamente cuando se oprime el botón, se hacen todos los cambios pertinentes una sola vez.

6. Ejemplo de aplicación: Cafetería

A fin de demostrar la utilidad de la biblioteca y la plantilla para crear nuevos proyectos, se procede a realizar una simulación utilizando únicamente los bloques que se han incluido en el trabajo. Esta simulación pretende ser una cafetería que acabe de incluir un sistema de control domótico y deba realizar la programación por primera vez.

6.1.- Especificaciones

Primero se define la instalación, estancias y sistemas susceptibles de poder ser automatizados de alguna forma. Para esta cafetería se ha considerado que consta de:

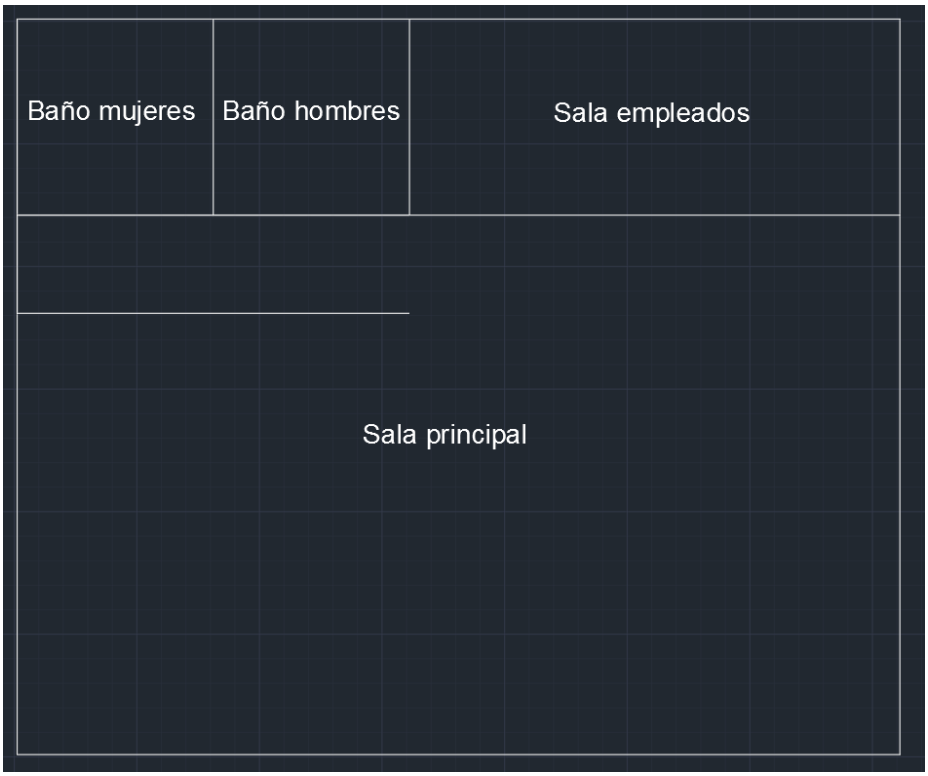

Figura 6-1 - Representación del local

- 1 sala principal para los clientes y barra.
- 1 sala para los empleados/almacenamiento/cocina.

- 1 baño para hombres.
- 1 baño para mujeres.
- 1 pequeña parte ajardinada en el exterior.
- 1 sistema de toldos para el exterior de la sala principal, todos respondiendo a la misma activación.
- 1 sistema de persiana metálica para el cierre.

Con esos datos se pretende incluir para esta simulación la totalidad de la capacidad de la plantilla, utilizando sus elementos de la siguiente forma:

- Luces:
    - 2 bloques de luces para los baños, en modo detector de presencia.
    - 1 bloque de luz para la sala principal, en modo regulación con sensor.
    - 1 bloque de luz para la sala de empleados/almacén/cocina, en modo interruptor.
- Alarmas: a incluir todo el conjunto.
- Riego: utilizar el bloque para la zona ajardinada en modo horario.
- Motores:
    - 1 bloque para el manejo de los toldos en modo automático.
    - 1 bloque para el manejo manual de las persianas de cierre.
- Climatización:
    - 1 bloque para la sala principal en modo horario.
    - 1 bloque para la sala de empleados/almacén/cocina en modo horario.

- Reloj: debido al uso de modos horarios es necesario incluirlo.

6.2.- Construcción del programa

Se comienza por crear en Codesys un nuevo proyecto llamado **Cafetería**, con el programa en texto estructurado, para aprovechar la plantilla.

El primer paso es añadir la biblioteca creada al proyecto desde el administrador, desde ahí Agregar biblioteca y en la categoría *Applications*, se encuentra con el nombre domótica.

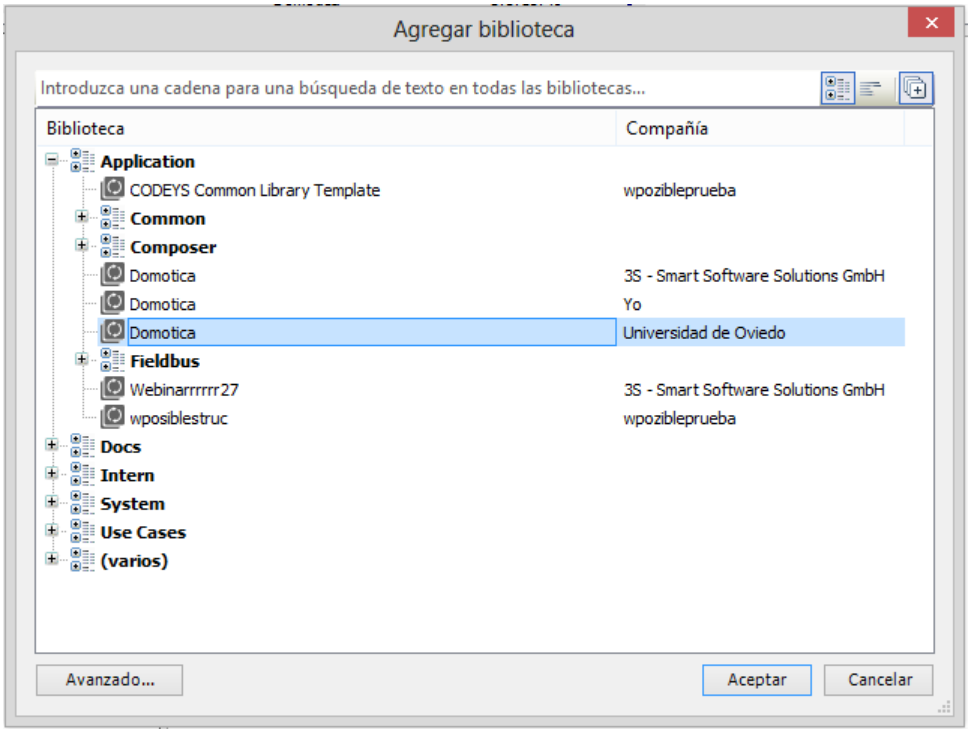

Figura 6-2 - Simulación incluyendo biblioteca

Se añade el archivo para las variables globales, el cual se nombra como GVL.

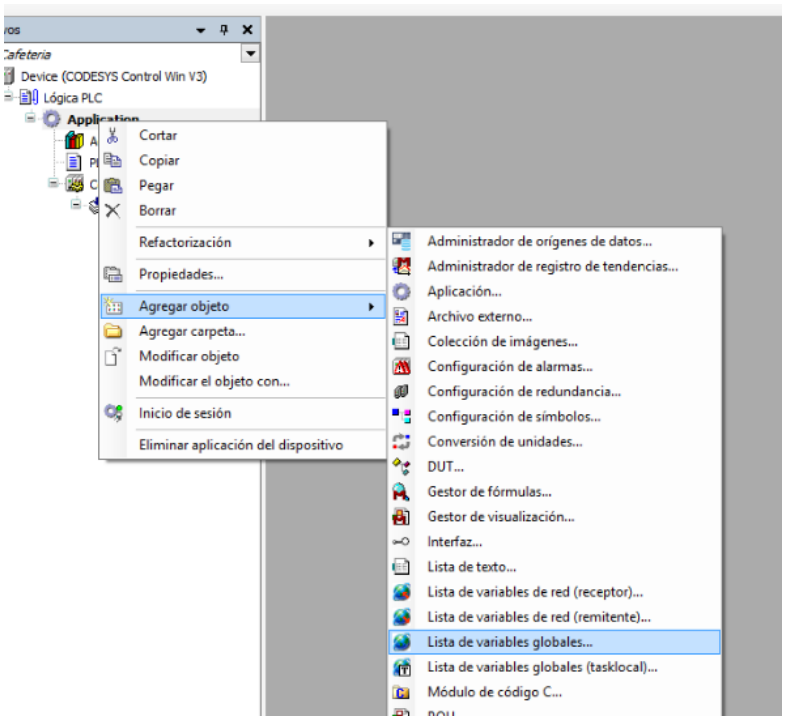

Figura 6-3 - Simulación Variables globales

El siguiente paso es utilizar la plantilla para generar una configuración viable, se introduce en la pestaña de configuración el nombre de variables globales, nombres para cada bloque según a que estancia o sistema hagan referencia y se eligen las características oportunas. Se utilizan los nombres de variables incluidos por defecto, y los valores iniciales correspondientes en la pestaña Plantilla.

Nombre de variables globales	GVL

Reloj		
Bloque para la obtención de hora y fecha que utilizaran los otros bloques en modo horario	Activación	Nombre para el bloque
	Mostrar	Reloj

Luces			Opciones		
Bloque para el manejo de cargas de luminarias con varias opciones de funcionamiento	Activación Bloque 1	Nombre para el bloque	Apagado	Regulación	Horario
	Mostrar	SalaP	☐ No usar	☑ Usar	☐ No usar
	Activación Bloque 2	Nombre para el bloque	Apagado	Regulación	Horario
	Mostrar	Cocina	☐ No usar	☐ No usar	☐ No usar
	Activación Bloque 3	Nombre para el bloque	Apagado	Regulación	Horario
	Mostrar	AseoM	☑ Usar	☐ No usar	☐ No usar
	Activación Bloque 4	Nombre para el bloque	Apagado	Regulación	Horario
	Mostrar	AseoH	☑ Usar	☐ No usar	☐ No usar

Figura 6-4 - Simulación extracto de la plantilla

Se generan los códigos y estos se incluyen en el proyecto.

```
VAR_GLOBAL

//BloqueReloj

O_datFecha_Actual_Reloj:DATE;
O_todHora_Actual_Reloj:TOD;

//BloqueSalaP

I_iModoLuz_SalaP:INT:=7;
I_xMandoLuz_SalaP:BOOL;
I_iIntensidad_SalaP:INT;
I_iSensorLuminosidad_SalaP:INT;
I_todHoraL1_SalaP:tod;
I_todHoraL2_SalaP:tod;
I_rPropL_SalaP:REAL:=0.5;
I_rIntL_SalaP:REAL:=0.1;
I_rDerL_SalaP:REAL:=0;
O_xConmutar1_SalaP:BOOL;
O_xConmutar2_SalaP:BOOL;
O_iLuminosidad_SalaP:INT;
O_sModoLuz_SalaP:STRING;
```

```
PROGRAM PLC_PRG
VAR

Reloj:Domotica.FB_Reloj;
SalaP:Domotica.FB_Control_de_Luces;
Cocina:Domotica.FB_Control_de_Luces;
AseoM:Domotica.FB_Control_de_Luces;
AseoH:Domotica.FB_Control_de_Luces;
Intrusion:Domotica.FB_Alarma_Intrusion;
Tecnicas:Domotica.FB_Alarma_Tecnica;
Config:Domotica.FB_Configuracion_de_Alarma;
```

```
Reloj   (
    Out_datFecha_Actual=>GVL.O_datFecha_Actual_Reloj,
    Out_todHora_Actual=>GVL.O_todHora_Actual_Reloj,
    );

SalaP   (
    In_iSel_Modo:=GVL.I_iModoLuz_SalaP,
    In_xMando:=GVL.I_xMandoLuz_SalaP,
    In_iIntensidadRegulador:=GVL.I_iIntensidad_SalaP,
    In_iSensorLuminosidad:=GVL.I_iSensorLuminosidad_SalaP,
    In_rPID_KP:=GVL.I_rPropL_SalaP,
    In_rPID_TN:=GVL.I_rIntL_SalaP,
    In_rPID_TV:=GVL.I_rDerL_SalaP,
    Out_xConmutar1=>GVL.O_xConmutar1_SalaP,
    Out_xConmutar2=>GVL.O_xConmutar2_SalaP,
    Out_iLuminosidad=>GVL.O_iLuminosidad_SalaP,
    Out_sMSGModo=>GVL.O_sModoLuz_SalaP,
    );
```

Figura 6-5 - Simulación códigos generados

En este punto al programa solo le faltaría definir las direcciones físicas de las variables que se vayan a utilizar en el PLC correspondiente y ya estaría listo para funcionar.

6.3.- Construcción de la simulación

Como no se dispone de una instalación real para probar todo lo que se ha incluido, se comprueba su correcto funcionamiento con la simulación, creando los comportamientos para los sensores de aquellos elementos que lo necesiten y aprovechar las visualizaciones incluidas en la biblioteca desarrollada.

6.3.1.- Simulación de sensores

Estos no requieren programación adicional, se trata únicamente de añadir en la visualización elementos para alterar las entradas de sensores de los diferentes bloques, de esta manera consiguiendo cambios como si de una instalación real se tratara.

Algunos de los sensores simulados son, por ejemplo, el sensor para los detectores de presencia, el sensor de lluvia, los detectores de las alarmas o el nivel de humedad en el suelo.

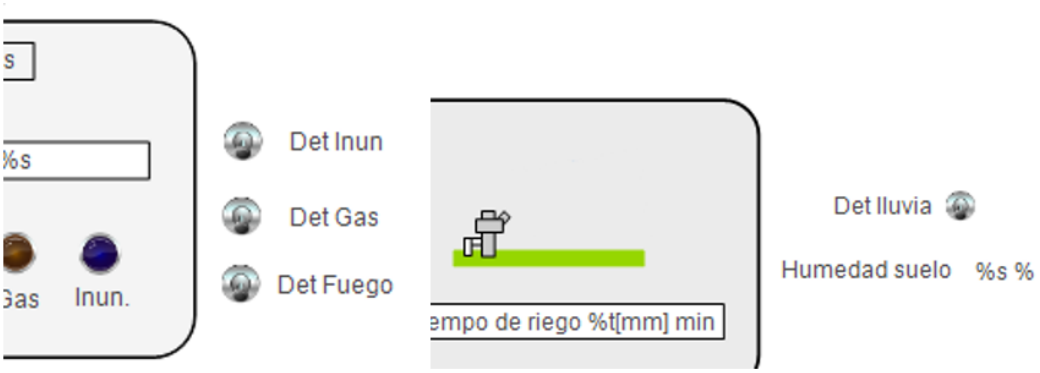

Figura 6-6 - Ejemplo de detectores

6.3.2.- Simulación de luminosidad

Al utilizar control por PID para los modos de regulación del bloque luces, se utiliza un sistema de primer orden para simular los cambios en la luminosidad de la estancia, dándole una mayor complejidad y realismo a los resultados.

La primera parte de la simulación está diseñada para el modo 6 del bloque de luces, regulación sin sensor de luminosidad.

```
1   //Simulación para luminosidad
2   //Simulacion de primer orden con driver de potencia 0-30000: Canal_Power (0-100%)
3
4   //Modo 6 regulacion sin sensor
5   IF GVL.I_iModoLuz_SalaP=6 THEN
6       LumModo6:=GVL.iLuzNatural+(GVL.O_iLuminosidad_SalaP*100/30000);
7
8       IF LumModo6<100 THEN
9       GVL.I_iSensorLuminosidad_SalaP:=LumModo6;
10      ELSE
11      GVL.I_iSensorLuminosidad_SalaP:=100;
12      END_IF
13  END_IF
```

Figura 6-7 - Simulación sin sensor

El código revisa el modo en el que se encuentra el bloque correspondiente, para saber si ejecutar o no la programación asociada.

Al no existir sensor en este modo, el nivel de luminosidad exigido en la entrada del bloque no pasa por regulación alguna y es directamente añadido a la cantidad de luz ambiente presente en la sala. Por último, se limita el valor que puede tomar la luminosidad para que no sobrepase el 100%, y se utiliza la propia variable de sensor de luminosidad, que en ese modo no tiene uso, para mostrar la luminosidad de la sala.

Si en cambio, se ha dispuesto el modo 7 que dispone de regulación con sensor, se utiliza la otra parte de la simulación, que llamara al sistema de primer orden para el comportamiento de la luminosidad.

```
16  //Modo 7 regulacion con sensor
17  //Transformacion de potencia dada por control de temperatura a entrada para el sistema de primer orden
18
19  IF GVL.I_iModoLuz_SalaP=7 THEN
20      IF GVL.O_xConmutar1_SalaP THEN
21          Canal_Power:=GVL.O_iLuminosidad_SalaP;
22      ELSE
23          Canal_Power:=0;
24      END_IF
25
26  //Llamada al sistema de primer orden, potencia y parametros
27      FIRST_ORDER_Luz ( UK:= Canal_Power, K:=K, T:=T, Tmms:=Tmms) ;
28  //Salida del primer orden a variacion de luminosidad
29      Canal_Luz := REAL_TO_INT ( FIRST_ORDER_Luz.YK );
30
31  //Límites superior e inferior para la variacion de luminosidad
32      IF Canal_Luz > 30000 THEN //Limite superior para la variacion de luminosidad
33          Canal_Luz := 30000;
34      END_IF
35
36      IF Canal_Luz <= 0 THEN //Limite inferior para la variacion de luminosidad
37          Canal_Luz := 0;
38      END_IF
39
40  //Transformacion necesaria para mostrar la luminosidad con decimales
41      rCanal_Luz := INT_TO_REAL (Canal_Luz);
42  //Modificacion de la luminosidad, luz narutal mas la variacion simulada
43      GVL.I_iSensorLuminosidad_SalaP:=GVL.iLuzNatural+(TO_INT(rCanal_Luz*100/30000));
44  END_IF
```

Figura 6-8 – Simulación de luminosidad regulada

Al igual que en el tramo anterior se comprueba en qué modo está el bloque asociado y se actúa o no en consecuencia.

Se llama al sistema de primer orden con los parámetros de constantes definidos para que actúe de una forma más o menos realista, y el nivel de luminosidad que esté ofreciendo el bloque asociado. El resultado que ofrece esta operación se transforma en la variación de luminosidad que habrá en la estancia, que se añade al nivel de luminosidad natural para dar un resultado que capta el sensor de luminosidad.

Este nuevo nivel de luminosidad se utiliza como referencia para el PID del bloque que intenta compensar los cambios producidos alterando su salida y todo el proceso se repite de nuevo hasta alcanzar un equilibrio.

```
VAR_INPUT
    UK: REAL; //Entrada
    K: REAL;  //Ganancia del sistema
    T:REAL;  //Constante de tiempo de sistema en segundos
    Tmms: REAL; //Periodo de muestreo en ms
END_VAR
VAR_OUTPUT
    YK: REAL;  //Salida respuesta del sistema
END_VAR
VAR
    YK_1: REAL; //Salida anterior
    Tm : REAL;
    a: REAL;
    b: REAL;
END_VAR
```

```
//Paso de periodo de muestreo a segundos

Tm := Tmms / 1000;

//ECUACIÓN:    YK := a * UK + b * YK_1

a := ( K * Tm ) /  (Tm + T ) ;
b :=  T /  (Tm + T ) ;

//Cálculo de la salida
YK :=a * UK + b * YK_1;

//Actualización del valor anterior
YK_1 := YK;
```

Figura 6-9 - Sistema de primer orden

6.3.3.- Simulación de temperatura

Al igual que en la simulación de luminosidad, también se realiza el comportamiento de la variación de temperatura a través de un sistema de primer orden, alterado sus parámetros para responder de una forma más lenta, pero lo bastante rápida para obtener resultados en unos pocos segundos.

Las simulaciones de temperatura de la sala principal y la sala de empleados comparten la misma estructura y valores, únicamente cambiando el nombre de las variables para cada una.

```
//Simulación para temperatura
//Sma de primer orden con KC y un supuesto driver de potencia: Canal_Power (0-100%)

//Transformacion de potencia daba al ambiente por control de temperatura a entrada para el sistema de primer orden
    IF GVL.O_xClimaBajaT_ClimaEmple THEN
        Canal_Power:=-GVL.O_rPotencia_Temp_ClimaEmple;
    ELSIF GVL.O_xClimaSubeT_ClimaEmple THEN
        Canal_Power:=GVL.O_rPotencia_Temp_ClimaEmple;
    ELSIF NOT GVL.O_xClimaBajaT_ClimaEmple AND NOT GVL.O_xClimaSubeT_ClimaEmple THEN
        Canal_Power:=0;
    END_IF

//Llamada al sistema de primer orden, potencia y parametros
    FIRST_ORDER_Temp ( UK:= Canal_Power, K:=K, T:=T, Tmms:=Tmms) ;
//Salida del primer orden a variacion de temperatura
    Canal_Temp := REAL_TO_INT ( FIRST_ORDER_Temp.YK * 100 );

//Límites superior e inferior para la variacion de temperatura
    IF Canal_Temp > 30000 THEN //Limite superior para la variacion de temperatura
        Canal_Temp := 30000;
    END_IF

    IF Canal_Temp <= -30000 THEN //Limite inferior para la variacion de temperatura
        Canal_Temp := -30000;
    END_IF

//Transformacion necesaria para mostrar la temperatura con decimales
    rCanal_Temp := INT_TO_REAL (Canal_Temp);
//Modificacion de la temperatura, tempeatura ambiente mas la variacion simulada +-15°C
    GVL.rTemp_Actual_ClimaEmple:=GVL.rTempAmbiente_ClimaEmple+(rCanal_Temp*15/30000);
```

Figura 6-10 - Simulación de temperatura

El bloque de temperaturas y su PID funcionan activando una de las salidas, subir o bajar temperatura, y dando una cantidad determinada de potencia. Debido a esto, y que esa potencia siempre será positiva para hacer funcionar el dispositivo correspondiente, se hace necesario invertir el sentido en el que varía la simulación cuando se activa la salida para disminuir la temperatura.

Esto se consigue con la primera parte del código, que asigna el valor correspondiente a la entrada del sistema de primer orden en función de la salida que se encuentre activa en ese momento, positiva si se aumenta la temperatura, negativa si ha de disminuir.

Mas adelante ese valor se utiliza como entrada para el sistema de primer orden, se evalúa su resultado dentro de unos límites establecidos, y la variable *rCanal_Temp* se utiliza como término que varía la temperatura respecto a la del ambiente a la que el sistema tiende de forma natural cuando no se le está suministrando potencia alguna.

La simulación está diseñada para conseguir una variación máxima de +-15ºC, de este modo combinándolo con que la visualización de temperaturas está limitada para temperaturas objetivo desde 15ºC a 30ºC, se puede conseguir todo ese rango de temperaturas.

6.4.- Visualizaciones

Para mostrar toda la simulación en funcionamiento se utilizan las diferentes visualizaciones incluidas en la biblioteca apoyándose en una pequeña representación del local.

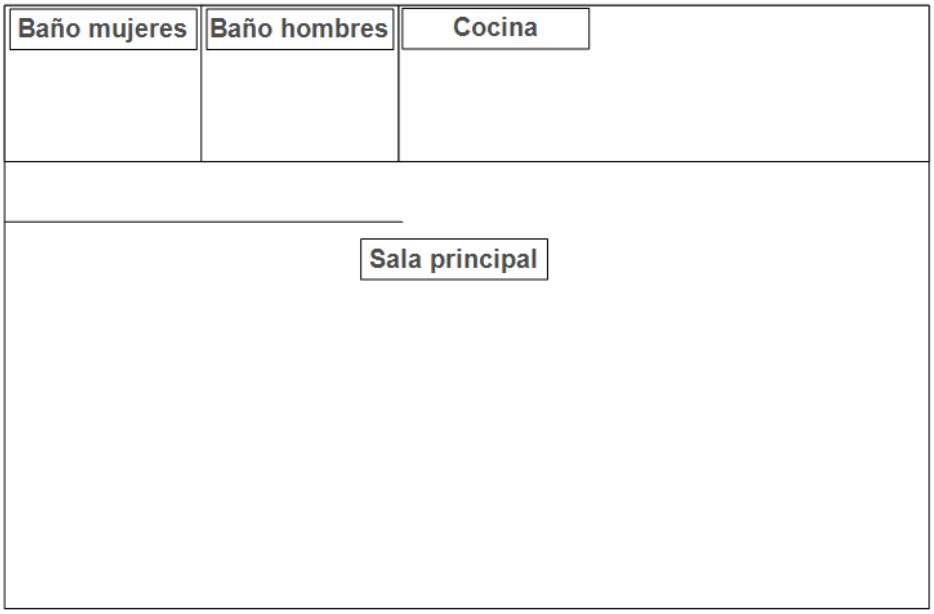

Figura 6-11 - Visualización cafetería

Cada una de las funciones principales; luces, motores, temperatura, alarmas y riego, cuentan con una visualización independiente en las que se encuentran sus controles para poder configurarse junto a los elementos que simulen sus entradas.

Junto a los controles para temperatura se incluye un gráfico en el que se puede observar la evolución de las diferentes variables a lo largo del tiempo. También se incluye un gráfico similar para la luz regulada.

Además, en cada una de las visualizaciones se incluyen unos botones adicionales para navegar entre pestañas con los que se puede cambiar fácilmente de una a otra.

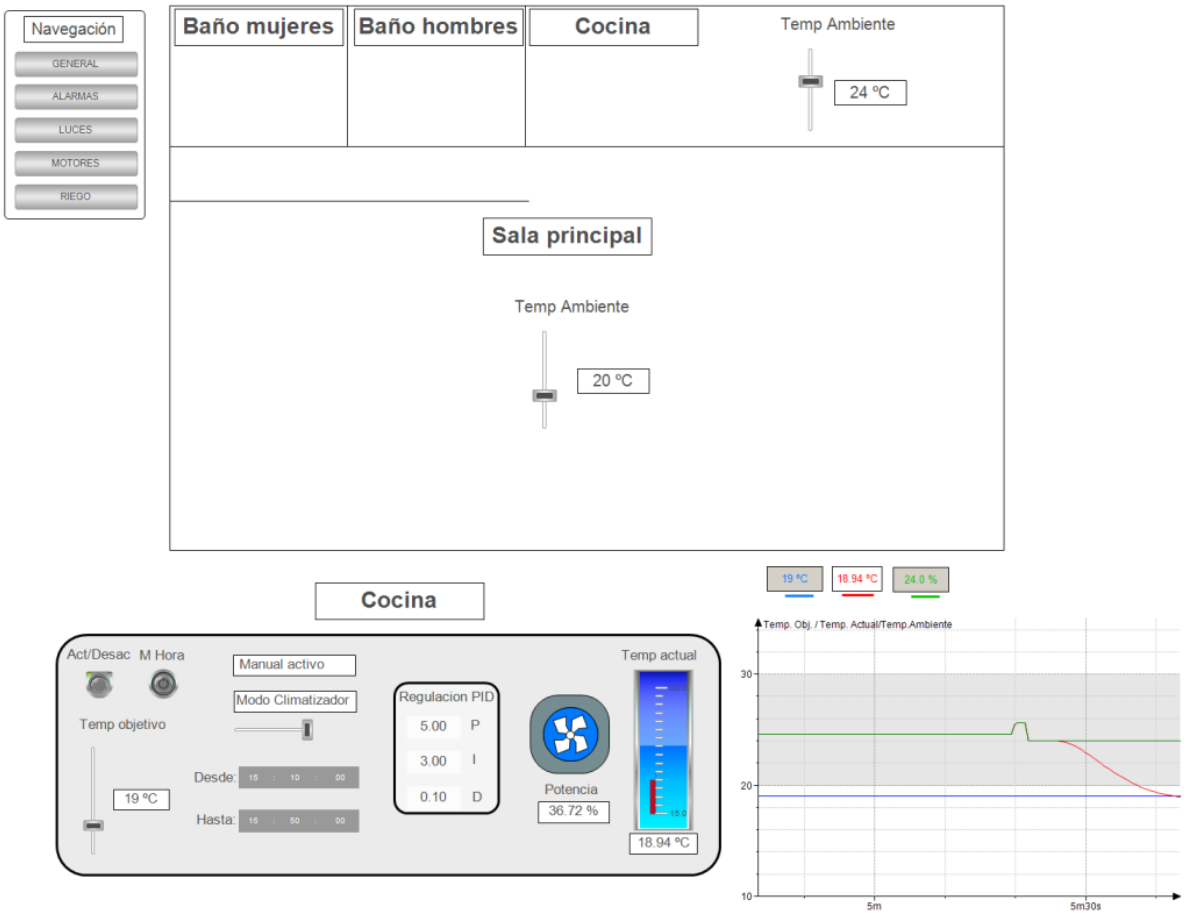

Figura 6-12 - Visualización de temperatura

Finalmente, utilizando las visualizaciones de elementos, se crea otra visualización adicional, nombrada como "VGeneral", que hace las veces de panel principal en la que se visualiza únicamente los diferentes elementos para poder tener una idea del estado de la instalación de un solo vistazo.

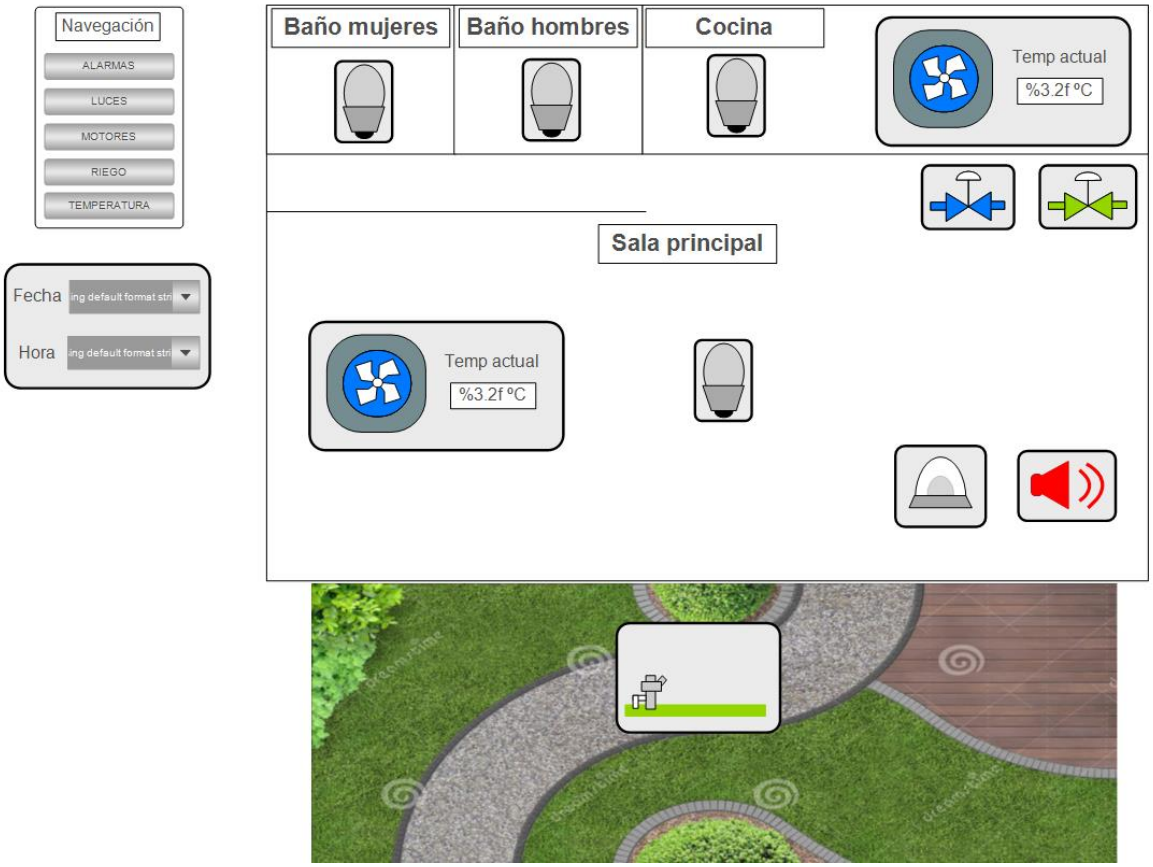

Figura 6-13 - Visualización "VGeneral"

7. Conclusiones y ampliaciones

Después de finalizar el proyecto se expone la impresión que deja todo el trabajo realizado por parte del autor.

Para llegar a los resultados obtenidos ha sido necesario el estudio de la domótica, los PLC, los estándares de programación y sus lenguajes, todos aspectos importantes y presentes en la programación y automatización de sistemas hoy en día. También ha sido necesario el familiarizarse y practicar con la herramienta software Codesys, la cual ha resultado ser muy potente, facilitando la realización del proyecto sin la necesidad de un controlador real ni una instalación física.

En el desarrollo del proyecto se observó que, al menos en la versión utilizada de Codesys, existe un fallo al intentar utilizar ciertas formas gráficas. Esto se produjo principalmente al intentar utilizar elipses para representar las gotas de agua en el riego, al agrupar toda la visualización como un solo bloque, aquellas elipses que se hubieran girado se comportaban de forma anómala al cambiar el nivel de zoom. Debido a esto se optó por utilizar los círculos en su lugar.

Un pequeño detalle adicional es el añadir valores iniciales en la plantilla. Aunque la mayoría de los tipos de variables han podido ser delimitadas con la validación de datos que incorpora Excel, otras incurren en problemas. Algún ejemplo son los números decimales que utilizan el punto "." en Codesys, pero la coma "," en Excel, o los formatos de fecha y hora que no se transmiten de forma correcta.

El resultado final ha sido conseguir desarrollar varias funciones típicas de la domótica y adaptarlas lo mejor posible a un uso universal mediante bloques funcionales IEC 61131-3, dejando el código en abierto para que se pueda seguir mejorando en el futuro.

Por último, se listan a continuación algunos aspectos que se podrían haber mejorado de las funciones pero que no se han podido incluir, bien por falta de tiempo o experiencia.

1. **Mas modos horarios:** incluir control horario en otras funciones como el control de motores o la activación programada de alarmas que no se han incluido por falta de tiempo.

2. **Rutina de error:** al igual que el bloque de alarma de intrusión comunica que el sensor de intrusos está activo antes de iniciarse y el bloque de temperatura comprueba que la temperatura varíe, se trataría de añadir más comprobaciones de error en otras funciones. Esto podría ir desde advertir de una mala programación de variables a malfuncionamiento de los sensores.

3. **Circuitos de E/S:** a excepción del bloque de luces que puede manejar 2 salidas en uno de sus modos, el resto de los bloques o bien solo reaccionan a una entrada o modifican una salida. La idea sería incluir opciones para que pudieran ser múltiples sin necesidad de incluir código adicional.

4. **Uso PLC real:** podría resultar de gran interés ampliar el trabajo para el uso de algún PLC real, preferiblemente basado en el estándar IEC 61131-3, como los disponibles en el laboratorio de la firma Phoenix Contact o ABB en combinación con maquetas que emulan el comportamiento de una vivienda automatizada.

8. Planificación

A fin de ofrecer una planificación orientativa de las tareas, se organizan lo mejor posible las distintas actividades desarrolladas a lo largo del tiempo que ha tomado realizar el proyecto. Al describirlas en los siguientes apartados toman un carácter secuencial, sin embargo, algunas pueden solaparse y realizarse al mismo tiempo.

El proyecto comienza en abril y se va desarrollando a lo largo de los meses siguientes hasta la convocatoria de septiembre, con las pertinentes reuniones de control con el tutor cada varias semanas.

8.1.- Tareas previas

El inicio del proyecto trata de conocer la idea que representa el trabajo que se debe desarrollar para marcar objetivos acordes al mismo. A través de varias reuniones con el tutor sobre la finalidad y alcance, estos se llegan a definir. Además, se realiza la primera búsqueda de información a fin de enfocar el trabajo.

8.2.- Programación en Codesys

Antes de iniciar la programación es necesario comprender el funcionamiento de la herramienta Codesys para poder utilizarla correctamente, por ello se dedica un tiempo a realizar ensayos, programas de ejemplo y visualizaciones como preparación previa.

El desarrollo posterior de los bloques funcionales esta solapado, pues se desarrollan todos más o menos al mismo tiempo y luego se amplían según necesidad o capacidad de programación. Aun así, la programación se puede dividir de acuerdo a las siguientes subtareas:

- Funcionalidades básicas.
- Modos automáticos.

- Modos horarios donde se incluye el reloj.
- Visualizaciones.
- Biblioteca.

8.3.- Plantilla Excel

Surge por la necesidad que aparece a lo largo del proyecto para ofrecer una forma sencilla de utilizar la biblioteca, para un usuario no experto en la programación con Codesys. Requiere también aprendizaje en el uso de macros para realizarla.

8.4.- Simulación de demostración

A falta de un medio físico para comprobar el correcto funcionamiento de la biblioteca y la plantilla, se utiliza el modo simulación que incorpora Codesys para su verificación

Además, se aprovecha el tiempo utilizado en el montaje de esta simulación para corregir pequeños errores que se puedan haber pasado por alto a lo largo de todo el proyecto.

8.5.- Documentación

Finalmente, se realiza toda la documentación relativa al desarrollo y resultado del proyecto, así como toda la información recopilada para llevarlo a cabo y como utilizarlo. La documentación la compone la memoria, presupuesto y anexos.

	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Tareas previas**	**21 días**	**mar 07/04/20**	**mar 05/05/20**	
2		Definir objetivos	3 días	mar 07/04/20	jue 09/04/20	
3		Busqueda de información	18 días	vie 10/04/20	mar 05/05/20	2
4		**Programación**	**35 días**	**mié 06/05/20**	**mar 23/06/20**	**1**
5		Funcionalidades básicas	14 días	mié 06/05/20	lun 25/05/20	
6		Modos automáticos	7 días	mar 26/05/20	mié 03/06/20	
7		Modos horarios	7 días	jue 04/06/20	vie 12/06/20	
8		Visualizaciones	20 días	mar 19/05/20	lun 15/06/20	
9		Biblioteca	16 días	mar 02/06/20	mar 23/06/20	
10		**Creación de plantilla excel**	**11 días**	**mié 24/06/20**	**lun 06/07/20**	**4**
11		Diseño de plantilla	6 días	mié 24/06/20	mar 30/06/20	
12		Desarrollo de macros	8 días	sáb 27/06/20	lun 06/07/20	
13		**Simulación**	**7 días**	**mié 08/07/20**	**jue 16/07/20**	**10**
14		Simulación	7 días	mié 08/07/20	jue 16/07/20	
15		**Documentación**	**31 días**	**vie 17/07/20**	**vie 28/08/20**	**13**
16		Documentación	31 días	vie 17/07/20	vie 28/08/20	

Figura 8-1 - Planificación de tareas

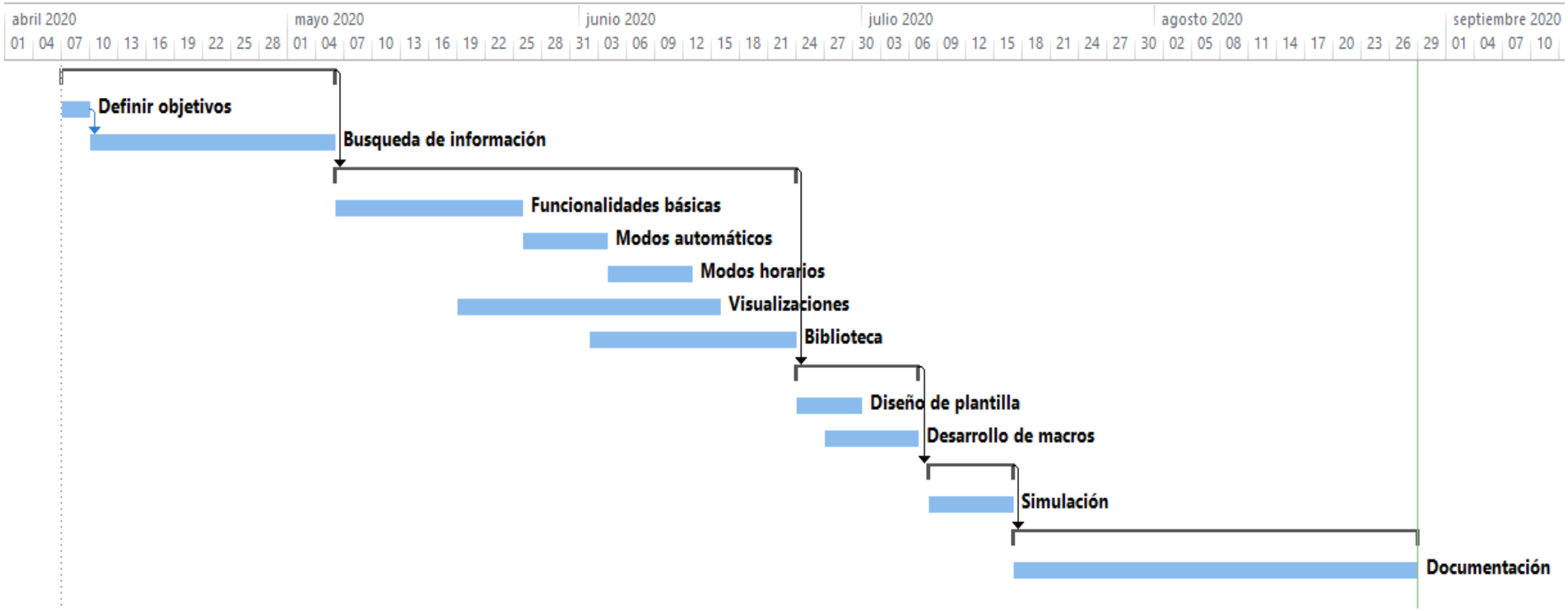

Figura 8-2 - Diagrama de Gant

9. Presupuesto

A continuación, se detalla el presupuesto total del proyecto.

Se desglosan todos los gastos que ha implicado la realización del TFM, incluyendo aprendizaje, documentación y demostración con simulación.

9.1.- Mediciones

9.1.1.- Software

Software			
Número	Concepto	Fabricante	Unidades
1	CODESYS V3.5 SP15 Patch 4	Codesys	1
2	Microsoft Office	Microsoft	1

Tabla 9-1 - Mediciones de software

9.1.2.- Mano de obra

9.1.2.1.- Labores previas

Labores previas			
Número	Concepto	Nº personas	Total h
1	Reuniones previas	1	3
2	Búsqueda de información	1	15
3	Estudio de documentación	1	20

Tabla 9-2 - Mediciones de labores previas

9.1.2.2.- Programación

Programación			
Número	Concepto	Nº personas	Total h
1	Familiarización con Codesys	1	20
2	Desarrollo de funciones	1	120
3	Elaboración de plantilla	1	18
4	Desarrollo de simulación	1	10

Tabla 9-3 - Mediciones de programación

9.1.2.3.- Documentación

Documentación			
Número	Concepto	Nº personas	Total h
1	Documentación para TFM	1	180

Tabla 9-4 - Mediciones de documentación

9.2.- Precios unitarios

9.2.1.- Software

Software				
Número	Concepto	Unidades	Precio ud (€)	Precio total (€)
1	CODESYS V3.5 SP15 Patch 4	Codesys	-	-
2	Microsoft Office	Microsoft	-	-
			TOTAL	-

Tabla 9-5 - Precios unitarios software

9.2.2.- Hardware

Software				
Número	Concepto	Unidades	Precio ud (€)	Precio total (€)
1	PC Portátil	5 meses	12,50 €	62,50 €
			TOTAL	62,50 €

Tabla 9-6 - Precios unitarios hardware

Se supone que un portátil cuesta 600 € y su duración es de 48 meses; el valor de amortización por mes es de 12,50 €, y si se usa 5 meses su coste al proyecto es de 62,50 €.

9.2.3.- Mano de obra

Para la mano de obra se tienen en cuenta los siguientes precios unitarios:

- 15 €/h para trabajos de búsqueda de información y documentación.
- 20 €/h para reuniones, trabajo de ingeniería y programación.

9.2.3.1.- Labores previas

Labores previas				
Número	Concepto	Total h	Coste h	Coste total
1	Reuniones previas	3	20€	60 €
2	Búsqueda de información	15	15 €	225 €
3	Estudio de documentación	20	15 €	300 €
			TOTAL	585€

Tabla 9-7- Precios unitarios labores previas

9.2.3.2.- Programación

Programación				
Número	Concepto	Total h	Coste h	Coste total
1	Familiarización con Codesys	20	15 €	300 €
2	Desarrollo de funciones	120	20 €	2.400 €
3	Elaboración de plantilla	18	20 €	360 €
4	Desarrollo de simulación	10	20 €	220 €
			TOTAL	3.280 €

Tabla 9-8- Precios unitarios programación

9.2.3.3.- Documentación

Documentación				
Número	Concepto	Total h	Coste h	Coste total
1	Documentación para TFM	180	15 €	2.700 €
			TOTAL	2.700 €

Tabla 9-9- Precios unitarios documentación

9.3.- Total

9.3.1.- Presupuesto de ejecución material

9.3.1.1.- Presupuesto total de materiales

Materiales	
Concepto	Presupuesto
Total software	-
Total hardware	62,50 €
Total Materiales	-

Tabla 9-10 - Presupuesto de la partida de materiales

9.3.1.2.- Presupuesto total de mano de obra

Mano de obra	
Concepto	Presupuesto
Labores previas	585 €
Programación	3.280 €
Documentación	2.700 €
Total Mano de obra	6.565 €

Tabla 9-11 - Presupuesto de la partida de mano de obra

9.3.1.3.- Presupuesto parcial

Ejecución material	
Concepto	Presupuesto
Partida de materiales (PC portátil, 5 meses)	62,50 €
Partida de mano de obra	6.565 €
Total Presupuesto de ejecución material	6.627,50 €

Tabla 9-12 - Presupuesto de ejecución material del proyecto

9.3.2.- Presupuesto por ejecución de contrata

9.3.2.1.- Presupuesto antes de impuestos

Presupuesto antes de impuestos	
Concepto	Presupuesto
Total ejecución material	6.627,50 €
Gastos generales (15%)	994,12 €
Beneficio industrial (6%)	397,65 €
Total antes de impuestos	8.019,27 €

Tabla 9-13 - Presupuesto antes de Impuestos

9.3.2.2.- Presupuesto por ejecución de contrata

Presupuesto de ejecución por contrata	
Concepto	Presupuesto
Total parcial	8.019,27 €
Iva (21%)	1.684,05 €
Total Presupuesto de ejecución por contrata	9.703,32 €

Tabla 9-14 - Presupuesto por ejecución de contrata

El presupuesto de ejecución por contrata de este TFM de la Escuela Politécnica de Ingeniería de Gijón asciende a NUEVE MIL SETECIENTOS TRES EUROS CON TREINTA Y DOS CÉNTIMOS, impuestos incluidos.

Fdo. Pedro Jesús Fernández Villanueva

Autor del Trabajo de Fin de Máster Diseño, desarrollo y validación de librería domótica IEC 61131-3.

10. Bibliografía

[1] Apuntes de la asignatura Domótica y edificios inteligentes, Oviedo de Universidad, 2020.

[2] Wikipedia, «Controlador lógico Programable,» 19 Abril 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Controlador_l%C3%B3gico_programable.

[3] Enfoque UTE, «Open Academic Journals Index,» 29 Abril 2020. [En línea]. Available: http://oaji.net/articles/2017/1783-1522269162.pdf.

[4] PLCopen, «IEC 61131: un resurso de programación estandar,» 24 Abril 2020. [En línea]. Available: http://isa.uniovi.es/docencia/ra_marina/cuatrim2/Temas/Intro_IEC_61131-3_Spanish.pdf.

[5] Siemens, «Tia Portal,» 2 Mayo 2020. [En línea]. Available: https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html.

[6] Unitronics, «Unilogic,» 3 Mayo 2020. [En línea]. Available: https://www.unitronicsplc.com/software-unilogic-for-programmable-controllers/.

[7] Codesys group, «Codesys,» 4 Mayo 2020. [En línea]. Available: https://www.codesys.com/.

[8] Codesys group, «Codesys Onlinne Help:Libraries,» 6 Mayo 2020. [En línea]. Available: https://help.codesys.com/webapp/f_libraries;product=codesys;version=3.5.16.0.

[9] «Programas de ejemplo de otros usuarios,» 9 Mayo 2020. [En línea]. Available: https://stackoverflow.com/.

[10] infoPLC, «Tutoriales en Codesys,» 8 Mayo 2020. [En línea]. Available: https://www.infoplc.net/descargas/42-codesys.

[11] Wikipedia, «Información general sobre domótica,» 13 Abril 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Dom%C3%B3tica.